DATABASE PROJECT REPORT

MOVIE IMDB RATING

- Sagar Panchal ● Saylee Raut ●Pugazharasu tamil Oli ● Keerthana Lakshmanan

Credentials

Userid : tamiloliweb

Password : alohomora

We declare that we have completed this assignment completely and entirely on our own,without any consultation with others. We have read the UAB Academic Honor Code and understand that any breach of the Honor Code may result in severe penalties.

We also declare that the following percentage distribution ***faithfully*** represents individualgroup members' contributions to the completion of the assignment.

Name	Overall Contribution(%)	Major work items completed by me	Signature or initials	Date
Sagar Panchal	25%		SP	12/03/2021
Saylee Raut	25%		SR	12/03/2021
Pugazharasu Tamil Oli	25%		PT	12/03/2021
Keerthana Lakshmanan	25%		KL	12/03/2021

A1. Application Background , Requirements (use Case), and assumptions

a) Application Background –

Every released movie in the database can receive a vote (from 1 to 5) and review from users. Votes cast are then summed together and shown as a combined IMDb rating. A viewer can only vote for one movie at a time and a user can view movie depending on categories. Movie can be sorted based on name, year, runtime and one movie can win award can categorized accordingly.

b) Requirements (Use Case)-

System should be capable of allowing user to login.

Allow user to give rating and review with login credentials.

Watch movie depending on category, awards.

c) Assumptions-

1.One user can give many ratings.

2.One movie can have many categories.

3. One movie can have many ratings.

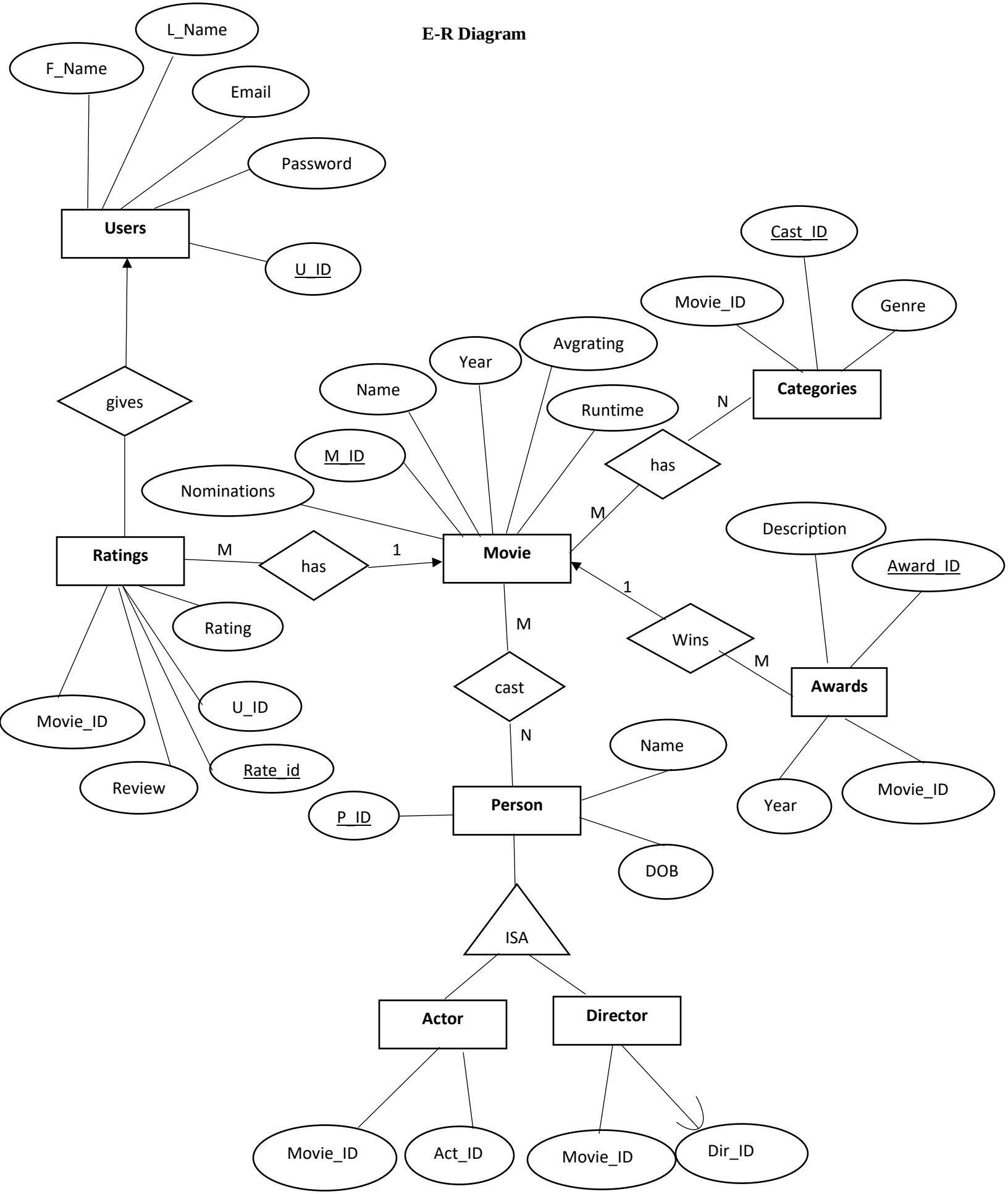
4. One award can win between many movies.

5.One category can have many movies and one movie can have many categories

6. Movie can cast many people who can be actor and directors.

7. People can be casted in one than more movies.

E-R Diagram


L_Name
F_Name
Email
Password
Users
U_ID
gives
Cast_ID
Movie_ID
Genre
Avgrating
Year
Name
Runtime
Categories
N
M_ID
has
Nominations
M
Description
Award_ID
Ratings
M
has
1
Movie
1
Rating
M
Wins
M
Awards
cast
U_ID
Movie_ID
N
Name
Rate_id
Review
P_ID
Person
Year
Movie_ID
DOB
ISA
Actor
Director
Movie_ID
Act_ID
Movie_ID
Dir_ID

ISA-

Actor is a person

Director is a person

Movie can have only one director and one movie can have many actors and actor can play role in many movies.

B. Relational Schema

Users (**U_id** , f_name, l_name, email, password)

Ratings (**rate_id**, move_id, review, u_id, rating)

Movie (**m_id**, Name, year, runtime, avgrating)

Categories (**Cast_id**, Move_id, Genre)

Awards (**Award_id**, Description, move_id, year)

People (**P_id**, Name, DOB)

Actor (M_id, Act_id)

Director (dir_id, M_id)

Explaination-

For table to be in 3NF it should satisfy 2nf and it should not have transitive dependency.

Actor and Director is redundant data which is separated from Person. As actor and director depends on person which is not a primary key. Which is said to be transitive dependency.

When there is attribute in a table which depends on some non-prime attribute.

So we took actor id and director id and put them in two sperate tables names Actor and director.

So its satisfy the condition ad now its in a 3rd NF.

Person-

P_id	Name	DOB	Act_id	D_Id

Person-

P_id	Name	DOB

Actor-

M_id	Act_id

Director-

M_id	Dir_id

C. Sample Data

MOVIE

```
CREATE TABLE movie (
   m_id integer PRIMARY KEY,
   name          varchar(20),
   year          integer,
   runtime       integer,
  avgrating      float,
 nominations     int
        );
```

```
tamiloli=> SELECT * FROM movie
tamiloli-> ;
 m_id |        name        | year | runtime | avgrating | nominations
------+--------------------+------+---------+-----------+-------------
  103 | Inception          | 2009 |     170 |         0 |           0
  104 | 3 Idiots           | 2009 |     170 |         0 |           0
  105 | Dark Knight        | 2008 |     152 |         0 |           0
  106 | Predestination     | 2014 |     107 |         0 |           0
  107 | The Prestige       | 2006 |     130 |         0 |           0
  108 | Mission Impossible | 2015 |     131 |         0 |           0
  110 | Avengers-4         | 2019 |     181 |         0 |           0
  109 | Avengers-3         | 2018 |     149 |         4 |           0
  101 | Eternal Sunshine   | 2008 |     108 |         0 |           2
  102 | Titanic            | 1997 |     185 |       4.1 |           0
(10 rows)
```

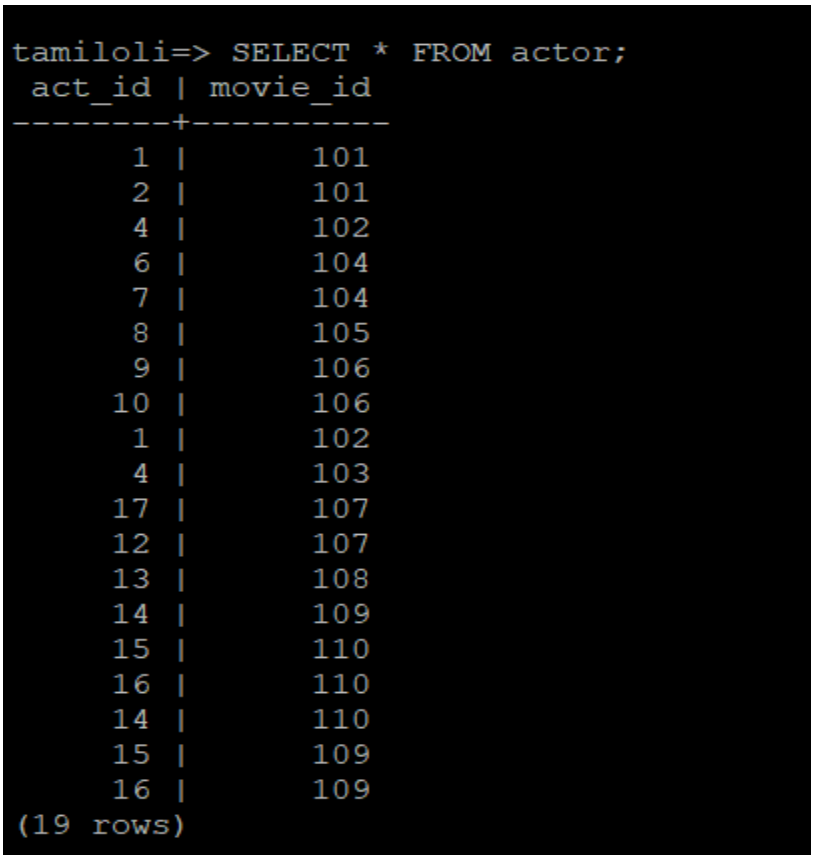
PERSON

```
CREATE TABLE person(
        p_id    integer  PRIMARY KEY,
        name    varchar(32) NOT NULL,
        dob     varchar(20) NOT NULL
);
```

```
tamiloli=> SELECT * FROM person;
 p_id |          name           |        dob
------+-------------------------+-------------------
    1 | Kate Winslet            | 5 Oct 1975
    2 | Jim Carry               | 17 Jan 1962
   11 | Michel Gondry           | 8 Nov 1973
    4 | Leonardo Di Caprio      | 11 Nov 1974
   22 | James Cameron           | 16 August 1954
   33 | Chris Nolan             | 30 July 1970
    6 | Christian Bale          | 30 January 1974
    7 | Heath Ledger            | 4 April 1979
    8 | Amir Khan               | 4 April 1979
   55 | Rajkumar Hirani         | 20 November 1962
    9 | Ethan Hawke             | 6 November 1970
   10 | Sara Snook              | 1 December 1987
   66 | Michael Spierig         | 29 April 1976
   17 | Hugh Jackman            | 12 Oct 1968
   12 | Chris Bale              | 30 January 1974
   13 | Tom Cruise              | 3 July 1962
   88 | Christopher McQuarrie   | 12 June 1968
   14 | Robert Downey Jr.       | 4 April 1965
   15 | Chris Evans             | 13 June 1981
   16 | Chris Hemsworth         | 11 August 1983
   99 | Joe Russo               | 18 July 1971
(21 rows)
```

ACTOR

```
CREATE TABLE actor(
        act_id          integer NOT NULL,
        movie_id        integer NOT NULL,
        CONSTRAINT fk_actor_person FOREIGN KEY(act_id) REFERENCES person(p_id)
);
```



```
tamiloli=> SELECT * FROM actor;
 act_id | movie_id
--------+----------
      1 |      101
      2 |      101
      4 |      102
      6 |      104
      7 |      104
      8 |      105
      9 |      106
     10 |      106
      1 |      102
      4 |      103
     17 |      107
     12 |      107
     13 |      108
     14 |      109
     15 |      110
     16 |      110
     14 |      110
     15 |      109
     16 |      109
(19 rows)
```

DIRECTOR

```
CREATE TABLE director(
        dir_id          integer  PRIMARY KEY,
        movie_id        integer NOT NULL,

        CONSTRAINT fk_director_person FOREIGN KEY(dir_id) REFERENCES person(p_id)
);
 INSERT INTO director VALUES(11,101);
```

```
tamiloli=> SELECT * FROM director;
 dir_id | movie_id
--------+----------
     11 |      101
     22 |      102
     33 |      103
     55 |      105
     66 |      106
     33 |      104
     33 |      107
     88 |      108
     99 |      109
     99 |      110
(10 rows)
```

CATEGORIES

CREATE TABLE categories (

cast_id integer PRIMARY KEY,

movie_id integer NOT NULL,

genre varchar(20)

CONSTRAINT fk_categories_movie FOREIGN KEY(movie_id) REFERENCES movie(m_id));

```
tamiloli=> select * from categories;
 cast_id | movie_id |  genre
---------+----------+----------
     201 |      101 | drama
     201 |      102 | drama
     202 |      101 | scifi
     202 |      104 | scifi
     203 |      101 | romance
     203 |      103 | romance
     204 |      105 | action
     204 |      106 | action
     205 |      106 | thriller
     205 |      107 | thriller
     206 |      107 | comedy
     206 |      108 | comedy
     207 |      109 | crime
     207 |      110 | crime
     201 |      150 | drama
(15 rows)
```

```
tamiloli=> \d categories;
         Table "public.categories"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 cast_id  | integer               | not null
 movie_id | integer               | not null
 genre    | character varying(20) |
Indexes:
    "pk" PRIMARY KEY, btree (cast_id, movie_id)
Foreign-key constraints:
    "fk_categories_movie" FOREIGN KEY (movie_id) REFERENCES movie(m_id)
```

AWARDS

CREATE TABLE awards (

award_id integer PRIMARY KEY,

movie_id integer NOT NULL,

CONSTRAINT fk_awards_movie FOREIGN KEY(award_id) REFERENCES movie(m_id));

```
tamiloli=> SELECT * FROM awards;
 award_id | movie_id |      description       | year
----------+----------+------------------------+------
      301 |      101 | Best Screenplay        | 2009
      302 |      102 | Best Visual Effects    | 1998
      303 |      103 | Best Screenplay        | 2010
      304 |      104 | Best Actor             | 2010
      305 |      105 | Best Supporting Actor  | 2009
      306 |      104 | Best Film              | 2009
      307 |      106 | Best BGM               | 2010
      308 |      108 | Best Stunt Chores      | 2016
      309 |      109 | Best CGI               | 2019
      310 |      110 | Best Screenplay        | 2020
(10 rows)
```

USERS

CREATE TABLE users (

u_id integer PRIMARY KEY,

f_name varchar(20),

l_name varchar(20),

email varchar(40) UNIQUE,

password varchar(40) NOT NULL

);

```
tamiloli=> SELECT * FROM users;
 u_id  |   f_name    |   l_name   |       email       |    password
-------+-------------+------------+-------------------+----------------
 10001 | Sagar       | Panchal    | sagar@uab.com     | qwerty
 10002 | Saylee      | Raut       | saylee@uab.com    | asdfgh
 10003 | Pugazharasu | Tamil Oli  | tamiloli@uab.com  | password
 10004 | Keerthana   | Lakshmanan | keerthana@uab.com | passworduab
 10005 | User        | Admin      | admin@uab.com     | uabengineering
(5 rows)
```

RATINGS

```
CREATE TABLE ratings (
   rate_id          integer PRIMARY KEY,
   movie_id         integer,
   u_id             integer NOT NULL,
   review           varchar(20),
   rating numeric CHECK (rating < 5),
CONSTRAINT fk_ratings_users FOREIGN KEY(u_id) REFERENCES users(u_id)
CONSTRAINT fk_ratings_movie FOREIGN KEY(movie_id) REFERENCES movie(m_id)
);
```

```
tamiloli=> SELECT * FROM ratings;
 rate_id | movie_id | u_id  |  review   | rating
---------+----------+-------+-----------+--------
     401 |      101 | 10001 | Good      |      4
     402 |      102 | 10003 | Excellent |    4.9
     403 |      102 | 10004 | average   |    3.5
     404 |      104 | 10002 | Good      |    4.5
     405 |      110 | 10001 | Average   |      3
     406 |      110 | 10002 | Average   |      3
     407 |      108 | 10003 | Good      |      4
     408 |      105 | 10004 | Average   |    2.8
     409 |      106 | 10002 | Excellent |    4.8
     410 |      109 | 10003 | Good      |    3.8
(10 rows)
```

D. Create Views

actorview

View to display the actor who has acted in the movie.

```
tamiloli=> Create view actorview as
tamiloli-> select a.act_id,p.name,m.name AS MovieName from person p, actor a, movie m
tamiloli-> where a.act_id=p.p_id and a.movie_id=m.m_id;
CREATE VIEW
tamiloli=> select * from actorview ;
 act_id |        name        |     moviename
--------+--------------------+-------------------
      1 | Kate Winslet       | Eternal Sunshine
      2 | Jim Carry          | Eternal Sunshine
      4 | Leonardo Di Caprio | Titanic
      6 | Christian Bale     | 3 Idiots
      7 | Heath Ledger       | 3 Idiots
      8 | Amir Khan          | Dark Knight
      9 | Ethan Hawke        | Predestination
     10 | Sara Snook         | Predestination
      1 | Kate Winslet       | Titanic
      4 | Leonardo Di Caprio | Inception
     17 | Hugh Jackman       | The Prestige
     12 | Chris Bale         | The Prestige
     13 | Tom Cruise         | Mission Impossible
     14 | Robert Downey Jr.  | Avengers-3
     15 | Chris Evans        | Avengers-4
     16 | Chris Hemsworth    | Avengers-4
     14 | Robert Downey Jr.  | Avengers-4
     15 | Chris Evans        | Avengers-3
     16 | Chris Hemsworth    | Avengers-3
(19 rows)
```

Directorview

View to display the name of the director who directed the movie

```
tamiloli=> Create view directorview as
select d.dir_id,p.name,m.name AS MovieName from person p, director d, movie m
where d.dir_id=p.p_id and d.movie_id=m.m_id;
CREATE VIEW
tamiloli=> select * from directorview ;
 dir_id |         name          |     moviename
--------+-----------------------+--------------------
     11 | Michel Gondry         | Eternal Sunshine
     22 | James Cameron         | Titanic
     33 | Chris Nolan           | Inception
     55 | Rajkumar Hirani       | Dark Knight
     66 | Michael Spierig       | Predestination
     33 | Chris Nolan           | 3 Idiots
     33 | Chris Nolan           | The Prestige
     88 | Christopher McQuarrie | Mission Impossible
     99 | Joe Russo             | Avengers-3
     99 | Joe Russo             | Avengers-4
(10 rows)
```

Awardsview

View to display the awards won by the movie.

```
tamiloli=> Create view awardsview as
tamiloli-> select m.name AS MovieName, A.description AS AwardWon, A.year
tamiloli-> from movie m, awards a
tamiloli-> where m.m_id=a.movie_id;
CREATE VIEW
tamiloli=> select * from awardsview ;
     moviename      |       awardwon        | year
--------------------+-----------------------+------
 Eternal Sunshine   | Best Screenplay       | 2009
 Titanic            | Best Visual Effects   | 1998
 Inception          | Best Screenplay       | 2010
 3 Idiots           | Best Actor            | 2010
 Dark Knight        | Best Supporting Actor | 2009
 3 Idiots           | Best Film             | 2009
 Predestination     | Best BGM              | 2010
 Mission Impossible | Best Stunt Chores     | 2016
 Avengers-3         | Best CGI              | 2019
 Avengers-4         | Best Screenplay       | 2020
(10 rows)

tamiloli=>
```

Movieratingview

View to display the movie rating for the movie.

```
tamiloli=> Create view movieratingview as
tamiloli-> select m.name AS MovieName, r.rating, r.review
tamiloli-> from Movie m, ratings r
tamiloli-> where m.m_id = r.movie_id;
CREATE VIEW
tamiloli=> select * from movieratingview ;
     moviename      | rating |  review
--------------------+--------+-----------
 Eternal Sunshine   |      4 | Good
 Titanic            |    3.5 | average
 Titanic            |    4.9 | Excellent
 3 Idiots           |    4.5 | Good
 Dark Knight        |    2.8 | Average
 Predestination     |    4.8 | Excellent
 Mission Impossible |      4 | Good
 Avengers-3         |    3.8 | Good
 Avengers-4         |      3 | Average
 Avengers-4         |      3 | Average
(10 rows)
```

avgratingview

View to display the average rating of the movie

```
tamiloli=> Create view avgratingview as
tamiloli-> select distinct m.name as MovieName, avg(rating)
tamiloli-> from movie m, ratings r
tamiloli-> where m.m_id = r.movie_id
tamiloli-> group by m.name;
CREATE VIEW
tamiloli=> select * from avgratingview ;
     moviename      |        avg
--------------------+--------------------
 Avengers-3         | 3.8000000000000000
 Titanic            | 4.2000000000000000
 Mission Impossible | 4.0000000000000000
 Predestination     | 4.8000000000000000
 Eternal Sunshine   | 4.0000000000000000
 Avengers-4         | 3.0000000000000000
 Dark Knight        | 2.8000000000000000
 3 Idiots           | 4.5000000000000000
(8 rows)
```

Moviedetailview

View to display the genre of the movie.

```
tamiloli=> Create view moviegenreview as
tamiloli-> select  c.genre, m.name
tamiloli-> from categories c, movie m
tamiloli-> where m.m_id = c.movie_id;
CREATE VIEW
tamiloli=> select * from moviegenreview ;
  genre   |        name
----------+--------------------
 drama    | Eternal Sunshine
 drama    | Titanic
 scifi    | Eternal Sunshine
 scifi    | 3 Idiots
 romance  | Eternal Sunshine
 romance  | Inception
 action   | Dark Knight
 action   | Predestination
 thriller | Predestination
 thriller | The Prestige
 comedy   | The Prestige
 comedy   | Mission Impossible
 crime    | Avengers-3
 crime    | Avengers-4
 drama    | UP
(15 rows)
```

E. Create Index

1. **Index 1: Index on director for dir_id to support directorview**

CREATE INDEX dir_id

ON director (dir_id);

```
tamiloli=> CREATE INDEX dir_id
ON director(dir_id);
CREATE INDEX
tamiloli=> \d director
    Table "public.director"
  Column  |  Type   | Modifiers
----------+---------+-----------
 dir_id   | integer | not null
 movie_id | integer | not null
Indexes:
    "dir_id" btree (dir_id)
Foreign-key constraints:
    "fk_director_person" FOREIGN KEY (dir_id) REFERENCES person(p_id)
```

2. **Index 2: Index on ratings for movie id and user id to support movierating view**

CREATE INDEX userrating

ON ratings (m_id,u_id);

```
tamiloli=> \d ratings
          Table "public.ratings"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 rate_id  | integer               | not null
 movie_id | integer               |
 u_id     | integer               | not null
 review   | character varying(20) |
 rating   | numeric               |
Indexes:
    "ratings_pkey" PRIMARY KEY, btree (rate_id)
    "userrating" btree (movie_id, u_id)
Check constraints:
    "ratings_rating_check" CHECK (rating < 5::numeric)
Foreign-key constraints:
    "fk_ratings_movie" FOREIGN KEY (movie_id) REFERENCES movie(m_id)
    "fk_ratings_users" FOREIGN KEY (u_id) REFERENCES users(u_id)
Triggers:
    avgratings AFTER INSERT OR DELETE OR UPDATE ON ratings FOR EACH ROW EXECUTE PROCEDURE process_avg_rating()
```

3. **Index 3 : Index on awards for award_id to support awardsview**
   CREATE INDEX award_id
   ON awards(award_id);

```
tamiloli=> CREATE INDEX award_id
ON awards(award_id);
CREATE INDEX
tamiloli=> \d awards
             Table "public.awards"
   Column    |         Type          | Modifiers
-------------+-----------------------+-----------
 award_id    | integer               | not null
 movie_id    | integer               | not null
 description | character varying(50) |
 year        | integer               |
Indexes:
    "awards_pkey" PRIMARY KEY, btree (award_id)
    "award_id" btree (award_id)
Triggers:
    nominations AFTER INSERT OR DELETE OR UPDATE ON awards FOR EACH ROW EXECUTE PROCEDURE process_nominations()
```

F.Constraints

```
tamiloli=> ALTER TABLE users
tamiloli-> ADD CONSTRAINT users_email_key UNIQUE(email);
ALTER TABLE
tamiloli=> \d users;
              Table "public.users"
  Column   |          Type          | Modifiers
-----------+------------------------+-----------
 u_id      | integer                | not null
 f_name    | character varying(20)  |
 l_name    | character varying(20)  |
 email     | character varying(40)  |
 password  | character varying(40)  | not null
Indexes:
    "users_pkey" PRIMARY KEY, btree (u_id)
    "users_email_key" UNIQUE CONSTRAINT, btree (email)
Referenced by:
    TABLE "ratings" CONSTRAINT "fk_ratings_users" FOREIGN KEY (u_id) REFERENCES users(u_id)

tamiloli=>
```

```
tamiloli=> \d movie
                  Table "public.movie"
   Column    |         Type          |     Modifiers
-------------+-----------------------+--------------------
 m_id        | integer               | not null
 name        | character varying(20) |
 year        | integer               |
 runtime     | double precision      |
 avgrating   | double precision      | not null default 0
 nominations | integer               | not null default 0
Indexes:
    "movie_pkey" PRIMARY KEY, btree (m_id)
    "year" btree (year)
Referenced by:
    TABLE "categories" CONSTRAINT "fk_categories_movie" FOREIGN KEY (movie_id) R
EFERENCES movie(m_id)
    TABLE "ratings" CONSTRAINT "fk_ratings_movie" FOREIGN KEY (movie_id) REFEREN
CES movie(m_id)
```

```
tamiloli=> ALTER TABLE users
tamiloli-> ADD CONSTRAINT unique_email UNIQUE (email);
ALTER TABLE
tamiloli=> \d users;
            Table "public.users"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 u_id     | integer               | not null
 f_name   | character varying(20) |
 l_name   | character varying(20) |
 email    | character varying(40) |
 password | character varying(40) | not null
Indexes:
    "users_pkey" PRIMARY KEY, btree (u_id)
    "unique_email" UNIQUE CONSTRAINT, btree (email)
    "users_email_key" UNIQUE CONSTRAINT, btree (email)
Referenced by:
    TABLE "ratings" CONSTRAINT "fk_ratings_users" FOREIGN KEY (u_id) REFERENCES users(u_id)

tamiloli=>
```

G. Triggers

Triggers 1

```
CREATE OR REPLACE FUNCTION process_nominations() RETURNS TRIGGER AS $nominations$
   BEGIN
      IF (TG_OP = 'INSERT') THEN
         UPDATE movie SET nominations = (SELECT count(*) FROM awards WHERE movie_id  = NEW.movie_id) WHERE movie.m_id = NEW.movie_id;
          ELSIF (TG_OP = 'UPDATE') THEN
         UPDATE movie SET nominations = (SELECT count(*) FROM awards WHERE movie_id  = OLD.movie_id) WHERE movie.m_id = OLD.movie_id;
          ELSIF (TG_OP = 'DELETE') THEN
         UPDATE movie SET nominations = (SELECT count(*) FROM awards WHERE movie_id  = OLD.movie_id) WHERE movie.m_id = OLD.movie_id;
      END IF;
      RETURN NULL;
   END;
$nominations$ LANGUAGE plpgsql;
```

```
tamiloli=> CREATE OR REPLACE FUNCTION process_nominations() RETURNS TRIGGER AS $nominations$
tamiloli$>     BEGIN
tamiloli$>         IF (TG_OP = 'INSERT') THEN
tamiloli$>             UPDATE movie SET nominations = (SELECT count(*) FROM awards WHERE movie_id  = NEW.movie_id)
tamiloli$>
ABORT           CHECKPOINT      COMMIT          DECLARE         DROP            FETCH           LOAD            PREPARE         RESET           SECURITY LABEL  START           UPDATE
ALTER           CLOSE           COPY            DELETE FROM     END             GRANT           LOCK            REASSIGN        REVOKE          SELECT          TABLE           VACUUM
ANALYZE         CLUSTER         CREATE          DISCARD         EXECUTE         INSERT          MOVE            REINDEX         ROLLBACK        SET             TRUNCATE        VALUES
BEGIN           COMMENT         DEALLOCATE      DO              EXPLAIN         LISTEN          NOTIFY          RELEASE         SAVEPOINT       SHOW            UNLISTEN        WITH
tamiloli$>
ABORT           CHECKPOINT      COMMIT          DECLARE         DROP            FETCH           LOAD            PREPARE         RESET           SECURITY LABEL  START           UPDATE
ALTER           CLOSE           COPY            DELETE FROM     END             GRANT           LOCK            REASSIGN        REVOKE          SELECT          TABLE           VACUUM
ANALYZE         CLUSTER         CREATE          DISCARD         EXECUTE         INSERT          MOVE            REINDEX         ROLLBACK        SET             TRUNCATE        VALUES
BEGIN           COMMENT         DEALLOCATE      DO              EXPLAIN         LISTEN          NOTIFY          RELEASE         SAVEPOINT       SHOW            UNLISTEN        WITH
tamiloli$> WHERE movie.m_id = NEW.movie_id;
tamiloli$> ELSIF (TG_OP = 'UPDATE') THEN
tamiloli$>             UPDATE movie SET nominations = (SELECT count(*) FROM awards WHERE movie_id  = NEW.movie_id)
tamiloli$>
ABORT           CHECKPOINT      COMMIT          DECLARE         DROP            FETCH           LOAD            PREPARE         RESET           SECURITY LABEL  START           UPDATE
ALTER           CLOSE           COPY            DELETE FROM     END             GRANT           LOCK            REASSIGN        REVOKE          SELECT          TABLE           VACUUM
ANALYZE         CLUSTER         CREATE          DISCARD         EXECUTE         INSERT          MOVE            REINDEX         ROLLBACK        SET             TRUNCATE        VALUES
BEGIN           COMMENT         DEALLOCATE      DO              EXPLAIN         LISTEN          NOTIFY          RELEASE         SAVEPOINT       SHOW            UNLISTEN        WITH
tamiloli$>
ABORT           CHECKPOINT      COMMIT          DECLARE         DROP            FETCH           LOAD            PREPARE         RESET           SECURITY LABEL  START           UPDATE
ALTER           CLOSE           COPY            DELETE FROM     END             GRANT           LOCK            REASSIGN        REVOKE          SELECT          TABLE           VACUUM
ANALYZE         CLUSTER         CREATE          DISCARD         EXECUTE         INSERT          MOVE            REINDEX         ROLLBACK        SET             TRUNCATE        VALUES
BEGIN           COMMENT         DEALLOCATE      DO              EXPLAIN         LISTEN          NOTIFY          RELEASE         SAVEPOINT       SHOW            UNLISTEN        WITH
tamiloli$> WHERE movie.m_id = NEW.movie_id;
tamiloli$> ELSIF (TG_OP = 'DELETE') THEN
tamiloli$>             UPDATE movie SET nominations = (SELECT count(*) FROM awards WHERE movie_id  = NEW.movie_id)
tamiloli$>
ABORT           CHECKPOINT      COMMIT          DECLARE         DROP            FETCH           LOAD            PREPARE         RESET           SECURITY LABEL  START           UPDATE
ALTER           CLOSE           COPY            DELETE FROM     END             GRANT           LOCK            REASSIGN        REVOKE          SELECT          TABLE           VACUUM
ANALYZE         CLUSTER         CREATE          DISCARD         EXECUTE         INSERT          MOVE            REINDEX         ROLLBACK        SET             TRUNCATE        VALUES
BEGIN           COMMENT         DEALLOCATE      DO              EXPLAIN         LISTEN          NOTIFY          RELEASE         SAVEPOINT       SHOW            UNLISTEN        WITH
tamiloli$>
ABORT           CHECKPOINT      COMMIT          DECLARE         DROP            FETCH           LOAD            PREPARE         RESET           SECURITY LABEL  START           UPDATE
ALTER           CLOSE           COPY            DELETE FROM     END             GRANT           LOCK            REASSIGN        REVOKE          SELECT          TABLE           VACUUM
ANALYZE         CLUSTER         CREATE          DISCARD         EXECUTE         INSERT          MOVE            REINDEX         ROLLBACK        SET             TRUNCATE        VALUES
BEGIN           COMMENT         DEALLOCATE      DO              EXPLAIN         LISTEN          NOTIFY          RELEASE         SAVEPOINT       SHOW            UNLISTEN        WITH
tamiloli$> WHERE movie.m_id = NEW.movie_id;
tamiloli$>         END IF;
tamiloli$>         RETURN NULL;
tamiloli$>     END;
tamiloli$> $nominations$ LANGUAGE plpgsql;
CREATE FUNCTION
```

CREATE TRIGGER nominations

AFTER INSERT OR UPDATE OR DELETE ON awards

FOR EACH ROW EXECUTE PROCEDURE process_nominations()

```
tamiloli=> CREATE TRIGGER nominations
tamiloli-> AFTER INSERT OR UPDATE OR DELETE ON awards
tamiloli->     FOR EACH ROW EXECUTE PROCEDURE process_nominations();
CREATE TRIGGER
tamiloli=>
```

INSERTING into awards table should reflect o movie table

```
tamiloli=> insert into awards values(312,101,'BEST ACTREESS',2009);
INSERT 0 1
tamiloli=> select * from movie;
 m_id |        name        | year | runtime | avgrating | nominations
------+--------------------+------+---------+-----------+-------------
  103 | Inception          | 2009 |     170 |         0 |           0
  104 | 3 Idiots           | 2009 |     170 |         0 |           0
  105 | Dark Knight        | 2008 |     152 |         0 |           0
  106 | Predestination     | 2014 |     107 |         0 |           0
  107 | The Prestige       | 2006 |     130 |         0 |           0
  108 | Mission Impossible | 2015 |     131 |         0 |           0
  110 | Avengers-4         | 2019 |     181 |         0 |           0
  102 | Titanic            | 1997 |     185 |         4 |           0
  109 | Avengers-3         | 2018 |     149 |         4 |           0
  101 | Eternal Sunshine   | 2008 |     108 |         0 |           3
(10 rows)
```

Here the Nominations column is updated after insertion.

DELETION:

```
tamiloli=> delete from awards where award_id = 312;
DELETE 1
tamiloli=> select * from movies;
ERROR:  relation "movies" does not exist
LINE 1: select * from movies;
                      ^
tamiloli=> select * from movie;
 m_id |        name        | year | runtime | avgrating | nominations
------+--------------------+------+---------+-----------+-------------
  103 | Inception          | 2009 |     170 |         0 |           0
  104 | 3 Idiots           | 2009 |     170 |         0 |           0
  105 | Dark Knight        | 2008 |     152 |         0 |           0
  106 | Predestination     | 2014 |     107 |         0 |           0
  107 | The Prestige       | 2006 |     130 |         0 |           0
  108 | Mission Impossible | 2015 |     131 |         0 |           0
  110 | Avengers-4         | 2019 |     181 |         0 |           0
  102 | Titanic            | 1997 |     185 |         4 |           0
  109 | Avengers-3         | 2018 |     149 |         4 |           0
  101 | Eternal Sunshine   | 2008 |     108 |         0 |           2
(10 rows)
```

The Nominations column is update after a deletion in Awards table.

Triggers 2

```
CREATE OR REPLACE FUNCTION process_avg_rating() RETURNS TRIGGER AS $avgRatings$
   BEGIN
      IF (TG_OP = 'INSERT') THEN
         UPDATE movie SET avgrating = (SELECT AVG(rating) FROM ratings WHERE movie_id  = NEW.movie_id) WHERE movie.m_id = NEW.movie_id;
      ELSIF (TG_OP = 'UPDATE') THEN
         UPDATE movie SET avgrating = (SELECT AVG(rating) FROM ratings WHERE movie_id  = OLD.movie_id) WHERE movie.m_id = OLD.movie_id;
      ELSIF (TG_OP = 'DELETE') THEN
         UPDATE movie SET avgrating = (SELECT AVG(rating) FROM ratings WHERE movie_id  = OLD.movie_id) WHERE movie.m_id = OLD.movie_id;
      END IF;
      RETURN NULL;
   END;
$avgRatings$ LANGUAGE plpgsql;


CREATE TRIGGER avgRatings
AFTER INSERT OR UPDATE OR DELETE ON ratings
   FOR EACH ROW EXECUTE PROCEDURE process_avg_rating();
```

Following is explanation for trigger **avgRating**.

Here, the user can insert/update/delete into the ratings table using **avgRating** trigger.

Whenever the trigger is called, the user inserts or updates or deletes the details from the ratings table. The **avgrating** column from the Movie table gets updated accordingly.

Following are example explanations performed for INSERT, UPDATE, DELETE.

CREATE TRIGGER FUNCTION:

```
moat.cs.uab.edu - PuTTY
tamiloli=> CREATE OR REPLACE FUNCTION process_avg_rating() RETURNS TRIGGER AS $avgRatings$
tamiloli$>     BEGIN
tamiloli$>         IF (TG_OP = 'INSERT') THEN
tamiloli$>             UPDATE movie SET avgrating = (SELECT AVG(rating) FROM ratings WHERE movie_id  = NEW.movie_id) WHERE movie.m_id = NEW.movie_id;
tamiloli$>
ABORT           CLOSE           CREATE          DO              FETCH           LOCK            REINDEX         SAVEPOINT       START           VACUUM
ALTER           CLUSTER         DEALLOCATE      DROP            GRANT           MOVE            RELEASE         SECURITY LABEL  TABLE           VALUES
ANALYZE         COMMENT         DECLARE         END             INSERT          NOTIFY          RESET           SELECT          TRUNCATE        WITH
BEGIN           COMMIT          DELETE FROM     EXECUTE         LISTEN          PREPARE         REVOKE          SET             UNLISTEN
CHECKPOINT      COPY            DISCARD         EXPLAIN         LOAD            REASSIGN        ROLLBACK        SHOW            UPDATE
tamiloli$> ELSIF (TG_OP = 'UPDATE') THEN
tamiloli$>             UPDATE movie SET avgrating = (SELECT AVG(rating) FROM ratings WHERE movie_id  = OLD.movie_id) WHERE movie.m_id = OLD.movie_id;
tamiloli$>
ABORT           CLOSE           CREATE          DO              FETCH           LOCK            REINDEX         SAVEPOINT       START           VACUUM
ALTER           CLUSTER         DEALLOCATE      DROP            GRANT           MOVE            RELEASE         SECURITY LABEL  TABLE           VALUES
ANALYZE         COMMENT         DECLARE         END             INSERT          NOTIFY          RESET           SELECT          TRUNCATE        WITH
BEGIN           COMMIT          DELETE FROM     EXECUTE         LISTEN          PREPARE         REVOKE          SET             UNLISTEN
CHECKPOINT      COPY            DISCARD         EXPLAIN         LOAD            REASSIGN        ROLLBACK        SHOW            UPDATE
tamiloli$> ELSIF (TG_OP = 'DELETE') THEN
tamiloli$>             UPDATE movie SET avgrating = (SELECT AVG(rating) FROM ratings WHERE movie_id  = OLD.movie_id) WHERE movie.m_id = OLD.movie_id;
tamiloli$>         END IF;
tamiloli$>         RETURN NULL;
tamiloli$>     END;
tamiloli$> $avgRatings$ LANGUAGE plpgsql;
CREATE FUNCTION
tamiloli=>
tamiloli=> CREATE TRIGGER avgRatings
tamiloli-> AFTER INSERT OR UPDATE OR DELETE ON ratings
tamiloli->     FOR EACH ROW EXECUTE PROCEDURE process_avg_rating();
CREATE TRIGGER
```

1. INSERT:
   i. Here we have inserted a new row in ratings table with rating = 4.
   ii. We can observe here that whenever an INSERT query is fired, the rating gets inserted in the movie table under avgrating.
   Here,
   avgrating from movie table for movie name Titanic is updated to 4.

```
tamiloli=> INSERT INTO ratings VALUES(411,102,10003,'Excellent',4);
INSERT 0 1
tamiloli=> select * from movie;
 m_id |        name        | year | runtime | avgrating | nominations
------+--------------------+------+---------+-----------+-------------
  103 | Inception          | 2009 |     170 |         0 |           0
  104 | 3 Idiots           | 2009 |     170 |         0 |           0
  105 | Dark Knight        | 2008 |     152 |         0 |           0
  106 | Predestination     | 2014 |     107 |         0 |           0
  107 | The Prestige       | 2006 |     130 |         0 |           0
  108 | Mission Impossible | 2015 |     131 |         0 |           0
  109 | Avengers-3         | 2018 |     149 |         0 |           0
  110 | Avengers-4         | 2019 |     181 |         0 |           0
  101 | Eternal Sunshine   | 2008 |     108 |         0 |           2
  102 | Titanic            | 1997 |     185 |         4 |           0
(10 rows)
```

2. Another record for movie id 102 is inserted in the ratings table with rating 3.
   We can see the avgrating from the movie table been updated.

```
tamiloli=> INSERT INTO ratings VALUES(415,102,10001,'Excellent',3);
INSERT 0 1
tamiloli=> SELECT * FROM MOVIE;
 m_id |        name        | year | runtime | avgrating | nominations
------+--------------------+------+---------+-----------+-------------
  103 | Inception          | 2009 |     170 |         0 |           0
  104 | 3 Idiots           | 2009 |     170 |         0 |           0
  105 | Dark Knight        | 2008 |     152 |         0 |           0
  106 | Predestination     | 2014 |     107 |         0 |           0
  107 | The Prestige       | 2006 |     130 |         0 |           0
  108 | Mission Impossible | 2015 |     131 |         0 |           0
  110 | Avengers-4         | 2019 |     181 |         0 |           0
  109 | Avengers-3         | 2018 |     149 |         4 |           0
  101 | Eternal Sunshine   | 2008 |     108 |         0 |           2
  102 | Titanic            | 1997 |     185 |      3.68 |           0
(10 rows)
```

3. Similarly, If we delete the rating id for that particular movie, the avgrating column from the movie table gets updated accordingly as shown below.

```
tamiloli=> delete from ratings where rate_id = 414;
DELETE 1
tamiloli=> SELECT * FROM MOVIE;
 m_id |        name        | year | runtime | avgrating | nominations
------+--------------------+------+---------+-----------+-------------
  103 | Inception          | 2009 |     170 |         0 |           0
  104 | 3 Idiots           | 2009 |     170 |         0 |           0
  105 | Dark Knight        | 2008 |     152 |         0 |           0
  106 | Predestination     | 2014 |     107 |         0 |           0
  107 | The Prestige       | 2006 |     130 |         0 |           0
  108 | Mission Impossible | 2015 |     131 |         0 |           0
  110 | Avengers-4         | 2019 |     181 |         0 |           0
  109 | Avengers-3         | 2018 |     149 |         4 |           0
  101 | Eternal Sunshine   | 2008 |     108 |         0 |           2
  102 | Titanic            | 1997 |     185 |      3.85 |           0
(10 rows)
```

4. When we try to update the rating for the existing rate_id from the rating id we observe the avgrating table getting updated accordingly.

```
tamiloli=> update ratings set rating = 4 where rate_id = 415;
UPDATE 1
tamiloli=> SELECT * FROM MOVIE;
 m_id |        name        | year | runtime | avgrating | nominations
------+--------------------+------+---------+-----------+-------------
  103 | Inception          | 2009 |     170 |         0 |           0
  104 | 3 Idiots           | 2009 |     170 |         0 |           0
  105 | Dark Knight        | 2008 |     152 |         0 |           0
  106 | Predestination     | 2014 |     107 |         0 |           0
  107 | The Prestige       | 2006 |     130 |         0 |           0
  108 | Mission Impossible | 2015 |     131 |         0 |           0
  110 | Avengers-4         | 2019 |     181 |         0 |           0
  109 | Avengers-3         | 2018 |     149 |         4 |           0
  101 | Eternal Sunshine   | 2008 |     108 |         0 |           2
  102 | Titanic            | 1997 |     185 |       4.1 |           0
(10 rows)
```

H. Stored Data

Stored Procedure to insert into movie table

Create OR REPLACE FUNCTION Add_movie (m_id INOUT INT, name varchar(100),year int,runtime float,avgrating float,nominations integer)

LANGUAGE plpgsql AS

$$ BEGIN

INSERT INTO movie Values (m_id,name,year,runtime,avgrating,nominations);

END $$;

```
tamiloli=>  Create OR REPLACE FUNCTION Add_movie (m_id INOUT INT, name varchar(100),year int,runtime float,avgrating float,nominations integer)
LANGUAGE plpgsql AS
$$ BEGIN

INSERT INTO movie Values (m_id,name,year,runtime,avgrating,nominations) ;

END $$;
CREATE FUNCTION
tamiloli=> SELECT Add_movie(150,'UP',2010,120,3,0);
 add_movie
-----------
       150
(1 row)

tamiloli=> select * from movie;
 m_id |        name        | year | runtime | avgrating | nominations
------+--------------------+------+---------+-----------+-------------
  103 | Inception          | 2009 |     170 |         0 |           0
  104 | 3 Idiots           | 2009 |     170 |         0 |           0
  105 | Dark Knight        | 2008 |     152 |         0 |           0
  106 | Predestination     | 2014 |     107 |         0 |           0
  107 | The Prestige       | 2006 |     130 |         0 |           0
  108 | Mission Impossible | 2015 |     131 |         0 |           0
  110 | Avengers-4         | 2019 |     181 |         0 |           0
  109 | Avengers-3         | 2018 |     149 |         4 |           0
  101 | Eternal Sunshine   | 2008 |     108 |         0 |           2
  102 | Titanic            | 1997 |     185 |       4.1 |           0
  150 | UP                 | 2010 |     120 |         3 |           0
(11 rows)
```