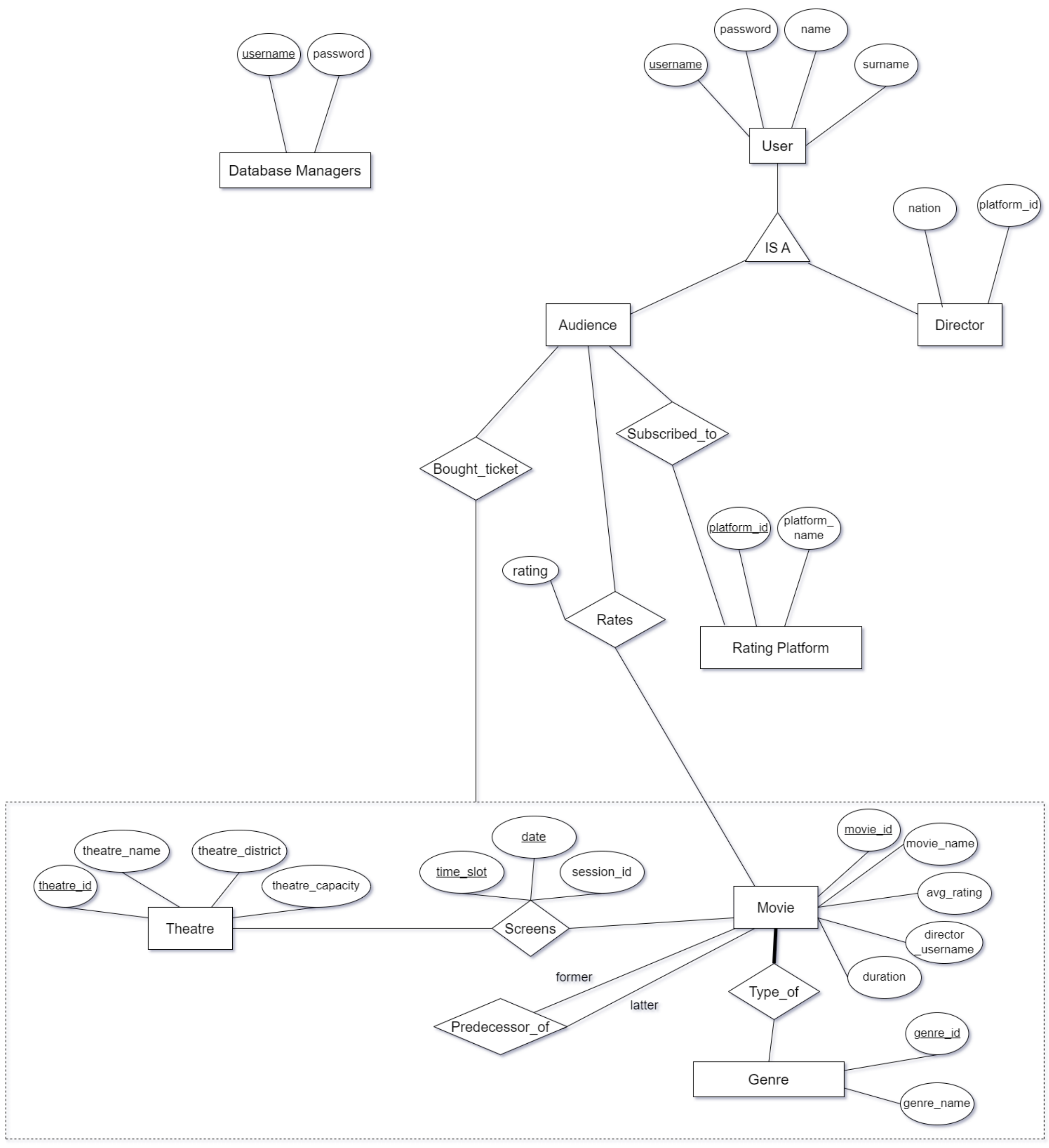
CMPE321 - Project 3: Movie DB

Muhammet Ali Topcu - 2020400147
Ömer Faruk Çelik - 2020400012

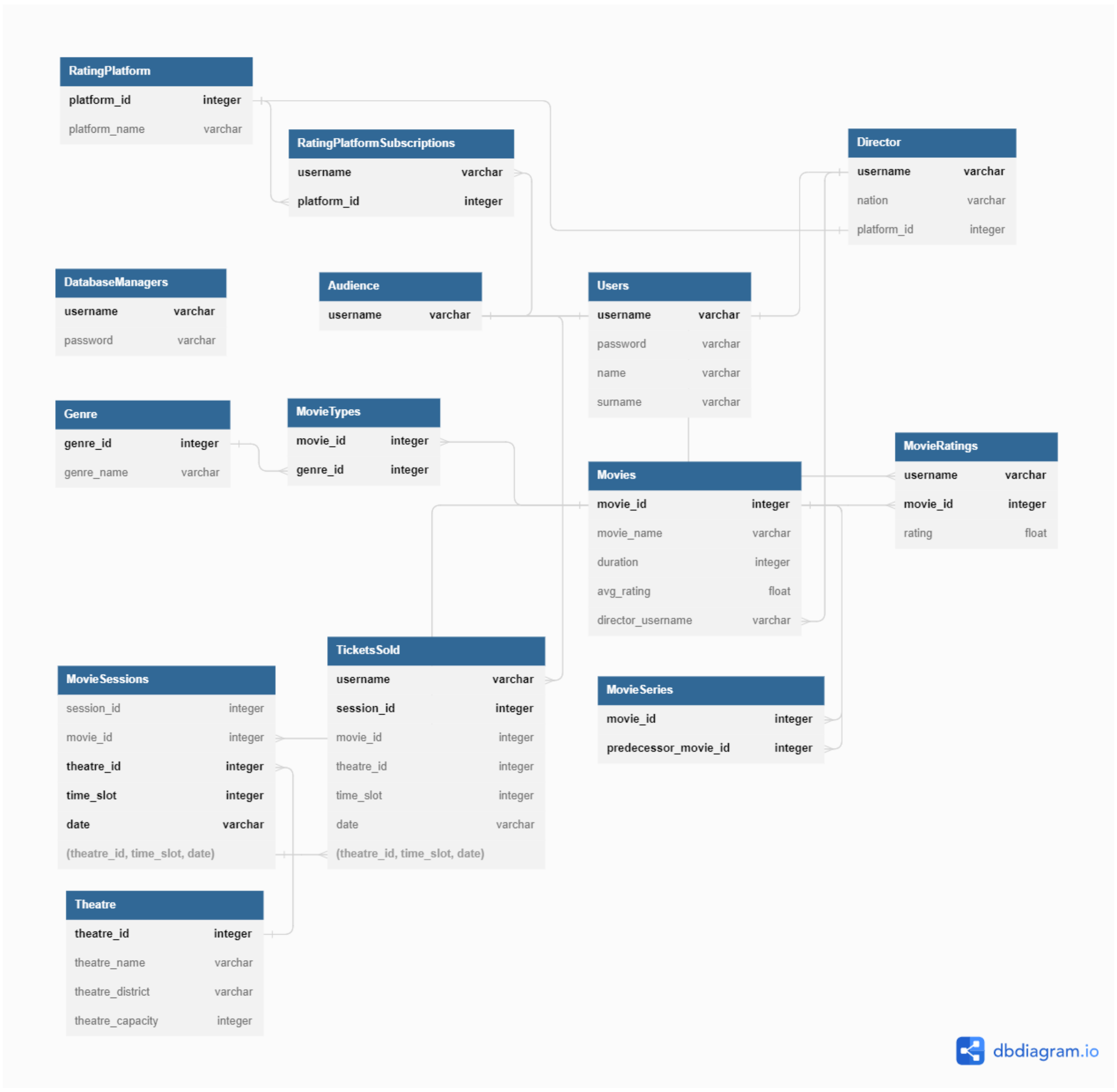
Project Description: Purpose of the project is refining ER diagram and database schemas for a booking and rating system for movies.

Design & Implementation

ER Diagram:



Logical Database Design:



Schemas:

Users(username: varchar, password: varchar, name: varchar, surname: varchar)

Audience(username: varchar)

RatingPlatform(platform_id: integer, platform_name: varchar)

Director(username: varchar, nation: varchar, platform_id: integer)

DatabaseManagers(username: varchar, password: varchar)

RatingPlatformSubscriptions(username: varchar, platform_id: integer)

Movies(movie_id: integer, movie_name: varchar, duration: integer, avg_rating: float, director_username: varchar)

MovieRatings(username: varchar, movie_id: integer, rationg: float)

Genre(genre_id: integer, genre_name: varchar)

MovieTypes(movie_id: integer, genre_id: integer)

MovieSeries(movie_id: integer, predecessor_movie_id: integer)

Theatre(theatre_id: integer, theatre_name: varchar, theatre_district: varchar, theatre_capacity: integer)

MovieSessions(session_id: integer, movie_id: integer, theatre_id: integer, time_slot: integer, date: varchar)

TicketsSold(username: varchar, session_id: integer, movie_id: integer, theatre_id: integer, time_slot: integer, date: varchar)

Schema Refinement Step:

USERS TABLE:
U -> U,P,N,S
Already in BCNF form since only FD is key constraint.

AUDIENCE TABLE:
No nontrivial BNCF. Already in BCNF form.

RATINGPLATFORM TABLE:
PID -> PID,PNAME
PNAME -> PID
Already in BCNF form. Since FDs are key constraint and PNAME is unique (So, a super key).

DIRECTOR TABLE:
U -> U,N,PID
Already in BCNF form since only FD is key constraint.

DATABASEMANAGERS TABLE:
U -> U,P
Already in BCNF form since only FD is key constraint.

RATINGPLATFORMSUBSCRIPTIONS TABLE:
No nontrivial BCNF. Already in BCNF form.

MOVIES TABLE:
MID -> MID, MNAME, D, AVGRATE, DNAME
Already in BCNF form since only FD is key constraint.

MOVIERATINGS TABLE:
U,MID -> U, MID, RATING
Already in BCNF form since only FD is key constraint.

GENRE TABLE:
GID -> GID, GNAME
GNAME -> GID
Already in BCNF form. Since FDs are key constraint and GNAME is unique (So, a super key).

MOVIETYPES TABLE:
No nontrivial BCNF. Already in BCNF form.

MOVIESERIES TABLE:
No nontrivial BCNF. Already in BCNF form.

THEATRE:
TID -> TID, TNAME, TDIST, TCAP
Already in BCNF form since only FD is key constraint.

MOVIESESSIONS TABLE:
TID, TSLOT, D -> SID, MID, TID, TSLOT, D
Already in BCNF form since only FD is key constraint.

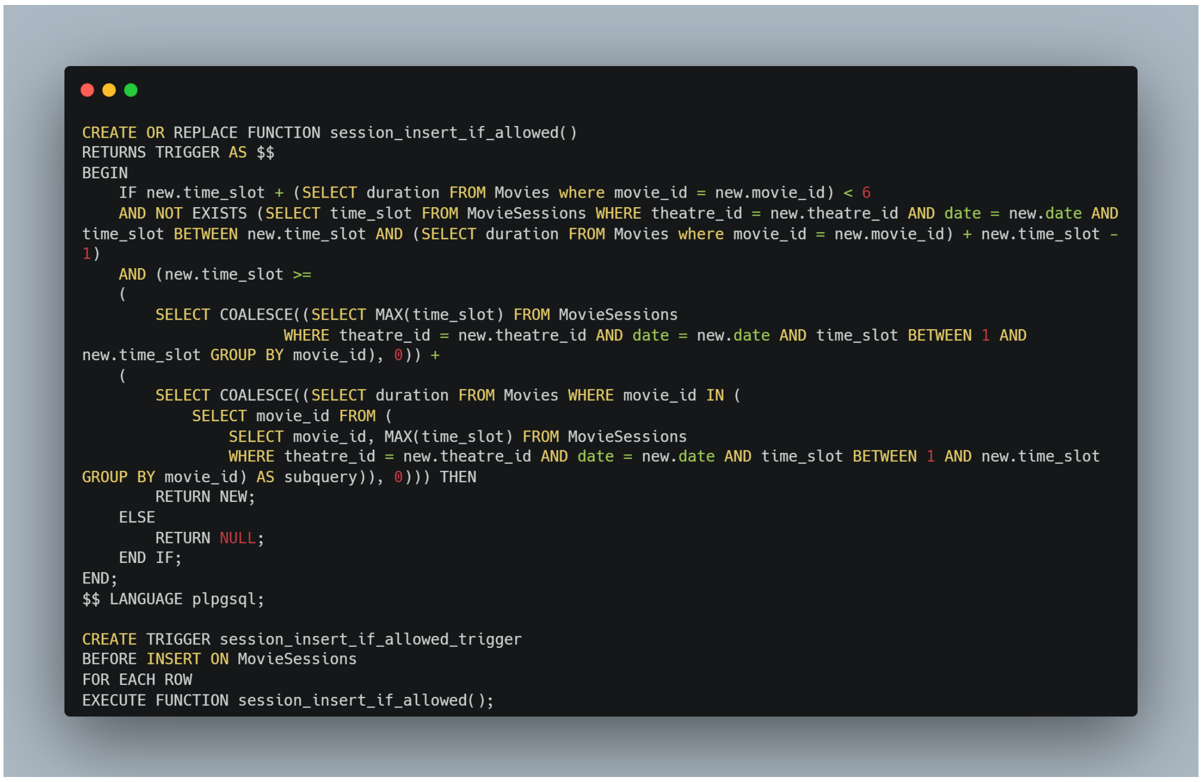
TICKETSSOLD TABLE:
U, SID -> U, SID, MID, TID, TSLOT
Already in BCNF form since only FD is key constraint.

Therefore, we didn't need to update ER diagram and Database schemas.

Capturing the Missing Constraints:

1. For these two constraints "No two movie sessions can overlap in terms of theatre and the time it's screened." and "The duration of the movie is closely related to the time slots. The time slot attribute determines the starting time of the movie and the end time is determined by the duration. (If a movie starts at time slot 2 and has a duration of 2, the theatre is reserved for that movie during the following time slots: [2, 3])", we need to address during insertion of movie sessions in our application layer. For example, before inserting a movie session, we need to first check whether there is an ongoing session at that (theatre_id, time_slot, date) usign the duration of the movie that are screened at that time slot.

We handled these constraints as follows:

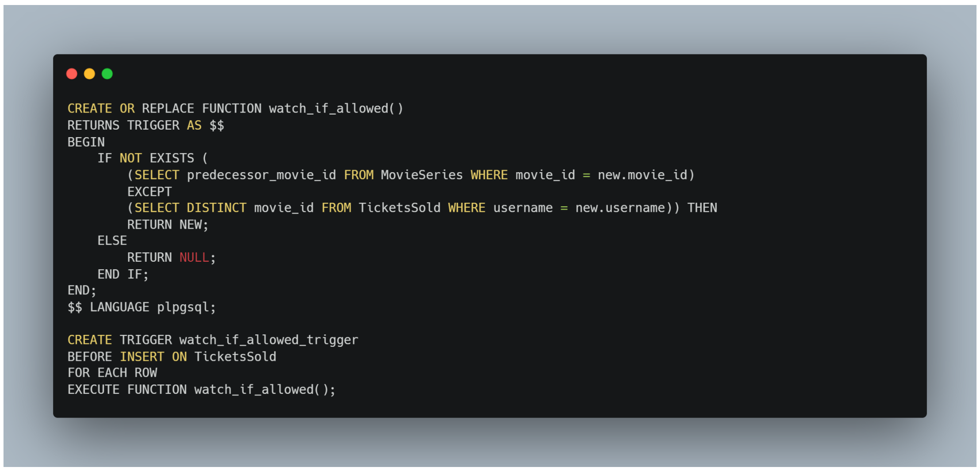


```
CREATE OR REPLACE FUNCTION session_insert_if_allowed()
RETURNS TRIGGER AS $$
BEGIN
    IF new.time_slot + (SELECT duration FROM Movies where movie_id = new.movie_id) < 6
    AND NOT EXISTS (SELECT time_slot FROM MovieSessions WHERE theatre_id = new.theatre_id AND date = new.date AND
time_slot BETWEEN new.time_slot AND (SELECT duration FROM Movies where movie_id = new.movie_id) + new.time_slot -
1)
    AND (new.time_slot >=
    (
        SELECT COALESCE((SELECT MAX(time_slot) FROM MovieSessions
                       WHERE theatre_id = new.theatre_id AND date = new.date AND time_slot BETWEEN 1 AND
new.time_slot GROUP BY movie_id), 0)) +
    (
        SELECT COALESCE((SELECT duration FROM Movies WHERE movie_id IN (
            SELECT movie_id FROM (
                SELECT movie_id, MAX(time_slot) FROM MovieSessions
                WHERE theatre_id = new.theatre_id AND date = new.date AND time_slot BETWEEN 1 AND new.time_slot
GROUP BY movie_id) AS subquery)), 0))) THEN
        RETURN NEW;
    ELSE
        RETURN NULL;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER session_insert_if_allowed_trigger
BEFORE INSERT ON MovieSessions
FOR EACH ROW
EXECUTE FUNCTION session_insert_if_allowed();
```

2. For the constraint "If a movie has any predecessor movies, all predecessor movies need to be watched in order to watch that movie. (See the example below: The Minions need to be watched before Minions: The Rise of Gru).", we need to address during insertion of a new ticket to TicketsSold. We need to check whether the audience watched all predecessor movies.

We handled this constraint as follows:

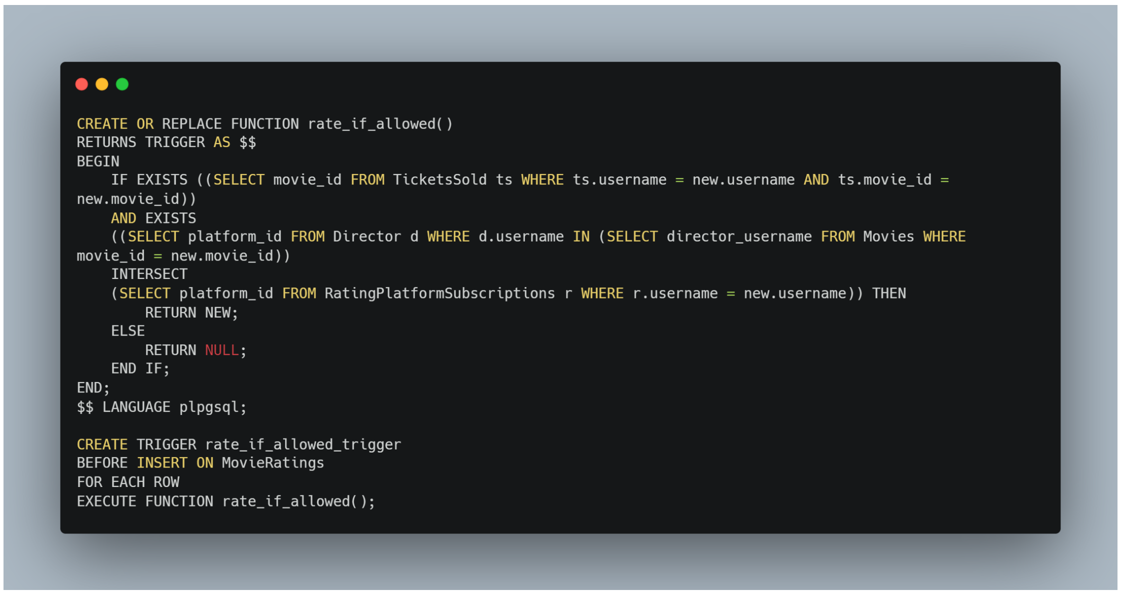


```
CREATE OR REPLACE FUNCTION watch_if_allowed()
RETURNS TRIGGER AS $$
BEGIN
    IF NOT EXISTS (
        (SELECT predecessor_movie_id FROM MovieSeries WHERE movie_id = new.movie_id)
        EXCEPT
        (SELECT DISTINCT movie_id FROM TicketsSold WHERE username = new.username)) THEN
        RETURN NEW;
    ELSE
        RETURN NULL;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER watch_if_allowed_trigger
BEFORE INSERT ON TicketsSold
FOR EACH ROW
EXECUTE FUNCTION watch_if_allowed();
```

3. For the constraint "A user can rate a movie if they are already subscribed to the platform that the movie can be rated AND if they have bought a ticket to the movie", we need to address during insertion of a rating to MovieRatings. We need to check whether the audience watched the movie and already bought a ticket for that movie.

We handled this constraints as follows:

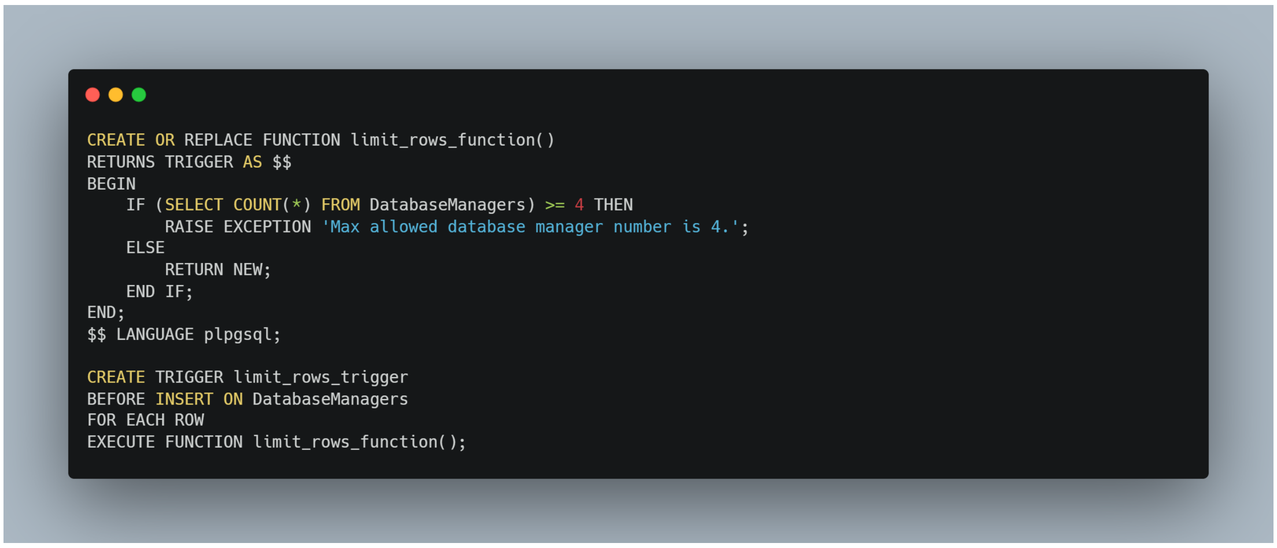


```
CREATE OR REPLACE FUNCTION rate_if_allowed()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS ((SELECT movie_id FROM TicketsSold ts WHERE ts.username = new.username AND ts.movie_id =
new.movie_id))
    AND EXISTS
    ((SELECT platform_id FROM Director d WHERE d.username IN (SELECT director_username FROM Movies WHERE
movie_id = new.movie_id))
    INTERSECT
    (SELECT platform_id FROM RatingPlatformSubscriptions r WHERE r.username = new.username)) THEN
        RETURN NEW;
    ELSE
        RETURN NULL;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER rate_if_allowed_trigger
BEFORE INSERT ON MovieRatings
FOR EACH ROW
EXECUTE FUNCTION rate_if_allowed();
```

4. For the constraint "There can be at most 4 database managers registered to the system.", we need to check the number of database managers in the system before inserting a new one in application layer.

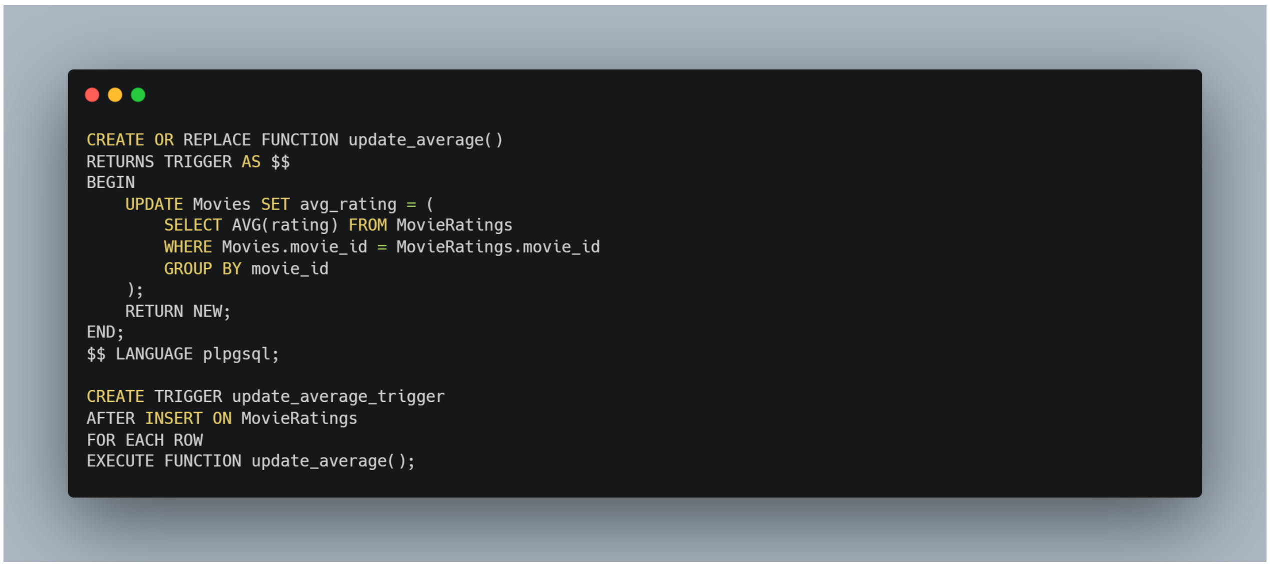
We handled this constraint as follows:



```
CREATE OR REPLACE FUNCTION limit_rows_function()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT COUNT(*) FROM DatabaseManagers) >= 4 THEN
        RAISE EXCEPTION 'Max allowed database manager number is 4.';
    ELSE
        RETURN NEW;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER limit_rows_trigger
BEFORE INSERT ON DatabaseManagers
FOR EACH ROW
EXECUTE FUNCTION limit_rows_function();
```

5. Additionally, we update overall ratings of movies with a trigger which can be seen below:



```
CREATE OR REPLACE FUNCTION update_average()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Movies SET avg_rating = (
        SELECT AVG(rating) FROM MovieRatings
        WHERE Movies.movie_id = MovieRatings.movie_id
        GROUP BY movie_id
    );
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_average_trigger
AFTER INSERT ON MovieRatings
FOR EACH ROW
EXECUTE FUNCTION update_average();
```