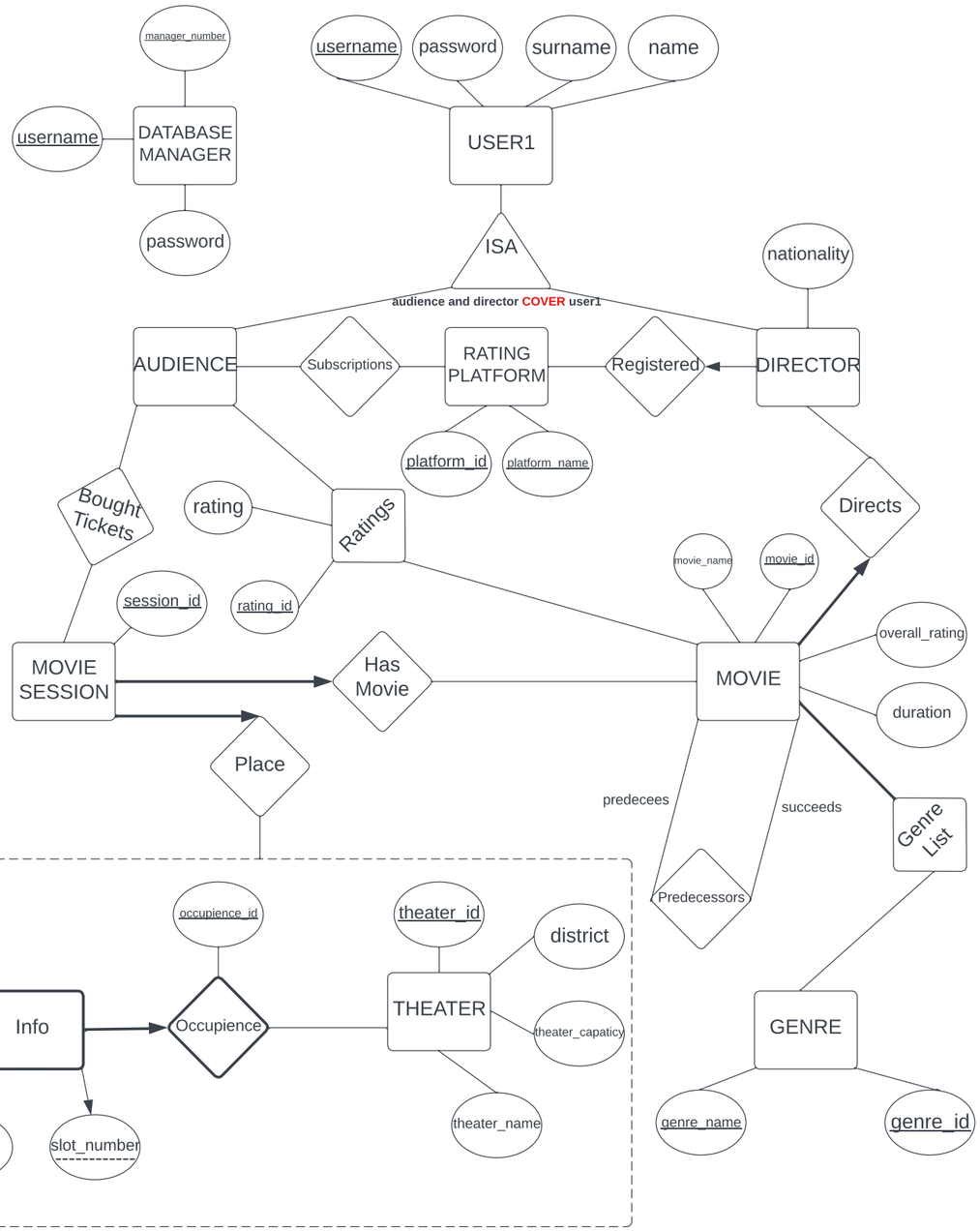


My implementation of the database & design approach:

1) The implementation of the *'Audience & Director'* is straightforward. They share the common attributes '*username*, *password*, *surname*, *name*'. Additionally, *director* has one more attribute, 'nationality' that is needed for the project requirements. (The constraint of each director **must** have **only one** nation is satisfied in the sql code with: direcor's username being PRIMARY KEY & nationality attribute NOT NULL )

2) The *rating platform* entity and its relations (*subscriptions & registered*) are also straightforward. *rating platform*'s *platform_id* ad *platform_name* are **both unique**. (This is satisfied in the sql code with: *platform_id* being PRIMARY KEY & *platform_name* being UNIQUE)

3) *Movie session* entity was the most challenging part of the project. I implemented the design focused on the reducing the redundacy caused by storing some attibutes more than once. Thus, *movie session* has only one attribute, which is also its key, *session_id*. *Movie session*'s other informations such as movie information and place & date & slot are stored in relations *'has movie'* and *'place'* consecutively. Since every *movie session* has a one movie, *movie session* entity **has a total participation & many to one** constraint in this relation. I used **aggregation** in *'place'* relation, because the inner relationship involves **weak entity**. Since every *movie session* has a **unique** *(theater_id, slot_number, time1), movie session* entity has **total participation and many to one** constraints in this relation. *Occupience* is a binary relation involving the weak entity *'Info'*. *Theater* entity is straightforward and has the entities described in the project. In conclusion, *place* relation is the key relation in *movie_session* entity. I used **aggregation** mainly because of the existence of the weak entity.

4) *Movie* entity is the second most complicated part of the project. Its attributes are straightforward and complies with the project requiremens. Naturally, *movie_id* is the primary key. *Movie* entity involves *directs* relation with the *director* entity with the constraints of **total participation & many to one**, satisfying the project requirements (Every movie needs to have **exactly one** director). There exist a unary relation called *Predecessors*, to track the movies that are needed to watch to the subjected movie. There also exists *Genre List* relation which acts as a list of genres that a movie possibly have. Movie entity has a **total participation** constraint because every movie have **at least one genre** according to project requirements.

5) There exists a relation called *ratings* and its main purpose is to keep track of the *audiences'* ratings to the movies.

6) Finally there exists a isolated entity called *'database manager'* for the managers of the database.

7) In order to reduce the storing redundant information I added *occupience_id* attribute to the *occupience_info* relation as primary key. Thus, when I use aggregation, I only need to store *occupience_id* attribute in the *place* relation.

8) I added *rating_id* attribute as primary key to the *ratings* relation because the tuple (*username*, *movie_id*) fails to become a primary key in the case when the *username* is NULL. (NULL *username* is allowed in *ratings* relation according to project description & discussion.)

The constraints that can not be handled with the diagrams:

1) 'nationality' attribute of the DIRECTOR entity **can not** be NULL. (This constraint will be satisfied in the sql code with the NOT NULL keyword.)

2) No two movie sessions **can not overlap** in terms of theatre and the time it's screened. (Starting times of the sessions can not overlap in the current diagram but the whole constraint requires some calculation with the MOVIE entity's 'duration' attribute, thus, can not be shown in the diagram.)

3) There are **four time slots** for each day. (This constraint will be satisfied in the SQL code with the CHECK keyword.)

4) If a movie has any predecessor movies, **all predecessor movies** need to be watched in order to watch the movie. (This constraint requires some advanced SQL code involving recursion and thus, can not be shown in the diagram.)

5) An user can rate a movie if they **already subscribed** to the platform that the movie can be rated AND if they **have bought a ticket** to the movie. (Constraint of platform match will be satisfied in the SQL code with rating_platforms_match TRIGGER. Constraint of have a bought ticket can also be satisfied in SQL code with another TRIGGER (not implemented).)

6) There can be **at most 4** database managers registered to the system. (This constraint will be satisfied in the SQL code with the CHECK.)

IMPORTANT NOTES:

1) The covering constraint of the ISA hierarchy is satisfied with a trigger that checks whenever inserting to the user1 table, relating username must be in the either director or audience table. Note that the existence of this trigger and the fact that director & audience has a column that references user1 table; makes adding a director or an audience to the audience/director table and user1 table impossible with sequential statements. Hence, one should use START TRANSACTION and COMMIT statements to avoid this issue. I would have avoided from this complexity with implementing the ISA hierarchy with just two entites of director and audience since we have the covering constraint. But then I thought; in most of the time, I would only need the username information of these entites (since it is the primary key) when I use the database and make some calculations in the sub-queires. So, I wanted to make these tables as minimal as possible with storing common attributes in the parent entity of user1.

2) When I name some entities/relations/attributes I added 1 as postfix, if they are keywords of the mysql without adding that postfix.

PREPARED BY: VOLKAN OZTURK 2019400033 & ARDA ARSLAN 2020400078

# ENTITY AND RELATION TABLES

PREPARED BY: VOLKAN OZTURK
2019400033 & ARDA ARSLAN
2020400078

USER1 (ENTITY)

| username (VARCHAR) | password (VARCHAR) | surname (VARCHAR) | name (VARCHAR) |
|---|---|---|---|
| steven.jobs | apple123 | Steven | Jobs |
| minion.lover | bello387 | Felonius | Gru |
| steve.wozniak | pass4321 | Ryan | Andrews |
| he.gongmin | passwordpass | He | Gongmin |
| carm.galian | madrid9897 | Carmelita | Galiano |
| kron.helene | helenepass | Helene | Kron |
| arzucan.ozgur | deneme123 | Arzucan | Ozgur |
| egemen.isguder | deneme124 | Egemen | Isguder |
| busra.oguzoglu | deneme125 | Busra | Oguzoglu |
| peter.weir | peter_weir879 | Peter | Weir |
| kyle.balda | mynameiskyle9 | Kyle | Balda |

DIRECTOR (ENTITY)

| username (VARCHAR) | nationality (VARCHAR) |
|---|---|
| he.gongmin | Turkish |
| carm.galian | Turkish |
| kron.helene | French |
| peter.weir | Spanish |
| kyle.balda | German |

AUDIENCE (ENTITY)

| username (VARCHAR) |
|---|
| steven.jobs |
| minion.lover |
| steve.wozniak |
| arzucan.ozgur |
| egemen.isguder |
| busra.oguzoglu |

DATABASE MANAGER (ENTITY)

| manager_number (INT) | username (VARCHAR) | password (VARCHAR) |
|---|---|---|
| 1 | manager1 | managerpass1 |
| 2 | manager2 | managerpass2 |
| 3 | manager3 | managerpass3 |
| 4 | manager4 | managerpass4 |

RATING PLATFORM (ENTITY)

| platform_id (INT) | platform_name (VARCHAR) |
|---|---|
| 10130 | IMDB |
| 10131 | Letterboxd |
| 10132 | FilmIzle |
| 10133 | Filmora |
| 10134 | BollywoodMDB |

MOVIE (ENTITY)

| movie_id (INT) | movie_name (VARCHAR) | overall_rating (INT) | duration (INT) |
|---|---|---|---|
| 20001 | Despicable Me | 5 | 2 |
| 20002 | Catch Me If You Can | NULL | 2 |
| 20003 | The Bone Collector | NULL | 2 |
| 20004 | Eagle Eye | 5 | 2 |
| 20005 | Minions: The Rise of Gru | 5 | 1 |
| 20006 | The Minions | 5 | 1 |
| 20007 | The Truman Show | 5 | 3 |

THEATER (ENTITY)

| theater_id (INT) | district (VARCHAR) | theater_capacity (INT) | theater_name (VARCHAR) |
|---|---|---|---|
| 40001 | Sisli | 300 | Sisli_1 |
| 40002 | Sisli | 200 | Sisli_2 |
| 40003 | Besiktas | 100 | Besiktas_1 |
| 40004 | Besiktas | 100 | Besiktas_2 |
| 40005 | Besiktas | 500 | Besiktas_3 |

GENRE (ENTITY)

| genre_id (INT) | genre_name (VARCHAR) |
|---|---|
| 80001 | Animation |
| 80002 | Comedy |
| 80003 | Adventure |
| 80004 | Real Story |
| 80005 | Thriller |
| 80006 | Drama |

MOVIE SESSION (ENTITY)

| session_id (INT) |
|---|
| 50001 |
| 50002 |
| 50003 |
| 50004 |
| 50005 |
| 50006 |
| 50007 |
| 50008 |
| 50009 |

SUBSCRIPTIONS (RELATION)

| username (VARCHAR) | platform_id (INT) |
|---|---|
| egemen.isguder | 10132 |
| arzucan.ozgur | 10130 |
| busra.oguzoglu | 10131 |
| steven.jobs | 10131 |
| steven.jobs | 10130 |
| steve.wozniak | 10131 |

REGISTERED (RELATION)

| username (VARCHAR) | platform_id (INT) |
|---|---|
| he.gongmin | 10130 |
| carm.galian | 10131 |
| kron.helene | 10130 |
| peter.weir | 10131 |
| kyle.balda | 10132 |

BOUGHT TICKETS (RELATION)

| username (VARCHAR) | session_id (INT) |
|---|---|
| egemen.isguder | 50008 |
| arzucan.ozgur | 50006 |
| busra.oguzoglu | 50009 |
| steven.jobs | 50001 |
| steve.wozniak | 50005 |
| steve.wozniak | 50004 |
| egemen.isguder | 50004 |
| egemen.isguder | 50007 |
| egemen.isguder | 50001 |

RATINGS (RELATION)

| username (VARCHAR) | movie_id (INT) | rating (INT) | rating_id (INT) |
|---|---|---|---|
| egemen.isguder | 20001 | 5 | 1 |
| egemen.isguder | 20005 | 5 | 2 |
| egemen.isguder | 20006 | 5 | 3 |
| arzucan.ozgur | 20004 | 5 | 4 |
| busra.oguzoglu | 20007 | 5 | 5 |

PREDECESSORS (RELATION)

| predecessor_movie_id (INT) | succeeds_movie_id (INT) |
|---|---|
| 20001 | 20006 |
| 20006 | 20005 |

PLACE (RELATION)

| session_id (INT) | occupience_id (INT) |
|---|---|
| 50001 | 1 |
| 50002 | 2 |
| 50003 | 3 |
| 50004 | 4 |
| 50005 | 5 |
| 50006 | 6 |
| 50007 | 7 |
| 50008 | 8 |
| 50009 | 9 |

DIRECTS (RELATION)

| movie_id (INT) | username (VARCHAR) |
|---|---|
| 20001 | kyle.balda |
| 20002 | he.gongmin |
| 20003 | carm.galian |
| 20004 | kron.helene |
| 20005 | kyle.balda |
| 20006 | kyle.balda |
| 20007 | peter.weir |

GENRE LIST (RELATION)

| movie_id (INT) | genre_id (INT) |
|---|---|
| 20001 | 80001 |
| 20001 | 80002 |
| 20002 | 80003 |
| 20002 | 80004 |
| 20003 | 80005 |
| 20004 | 80003 |
| 20005 | 80001 |
| 20005 | 80002 |
| 20006 | 80001 |
| 20006 | 80002 |
| 20007 | 80002 |
| 20007 | 80006 |

OCCUPIENCE INFO (RELATION)

| theater_id (INT) | slot_number (INT) | time1 (VARCHAR) | occupience_id (INT) |
|---|---|---|---|
| 40001 | 1 | 3/15/23 | 1 |
| 40001 | 3 | 3/15/23 | 2 |
| 40002 | 1 | 3/15/23 | 3 |
| 40002 | 3 | 3/15/23 | 4 |
| 40003 | 1 | 3/16/23 | 5 |
| 40003 | 3 | 3/16/23 | 6 |
| 40004 | 1 | 3/16/23 | 7 |
| 40004 | 3 | 3/16/23 | 8 |
| 40005 | 1 | 3/16/23 | 9 |

HAS MOVIE (RELATION)

| session_id (INT) | movie_id (INT) |
|---|---|
| 50001 | 20001 |
| 50002 | 20001 |
| 50003 | 20001 |
| 50004 | 20002 |
| 50005 | 20003 |
| 50006 | 20004 |
| 50007 | 20005 |
| 50008 | 20006 |
| 50009 | 20007 |

# SCHEMA REFINEMENT - DESIGN & CONSTRAINTS

**General Information About the Updated Database Design:**

Since our design is already in BCNF (the formal proof of this is in a seperate pdf file) hence, our ER diagrams remains same. However there are some changes in '*createTables.sql*' that sould be mentioned:

1) We decided to use PostgreSQL instead of MySQL. Thus we were needed to make some modifications related with some syntactic differences. Note that this does not change **anything effectively**.

2) The implementation & modification of the constraints would be compared and explained in the below.

PROJECT-1

The constraints that can not be handled with the diagrams:

1) 'nationality' attribute of the DIRECTOR entity **can not** be NULL. (This constraint will be satisfied in the sql code with the NOT NULL keyword.)

2) No two movie sessions **can not overlap** in terms of theatre and the time it's screened. (Starting times of the sessions can not overlap in the current diagram but the whole constraint requires some calculation with the MOVIE entity's 'duration' attribute, thus, can not be shown in the diagram.)

3) There are **four time slots** for each day. (This constraint will be satisfied in the SQL code with the CHECK keyword.)

4) If a movie has any predecessor movies, **all predecessor movies** need to be watched in order to watch the movie. (This constraint requires some advanced SQL code involving recursion and thus, can not be shown in the diagram.)

5) An user can rate a movie if they **already subscribed** to the platform that the movie can be rated AND if they **have bought a ticket** to the movie. (Constraint of platform match will be satisfied in the SQL code with rating_platforms_match TRIGGER. Constraint of have a bought ticket can also be satisfied in SQL code with another TRIGGER (not implemented).)

6) There can be **at most 4** database managers registered to the system. (This constraint will be satisfied in the SQL code with the CHECK.)

IMPORTANT NOTES:

1) The covering constraint of the ISA hierarchy is satisfied with a trigger that checks whenever inserting to the user1 table, relating username must be in the either director or audience table. Note that the existence of this trigger and the fact that director & audience has a column that references user1 table; makes adding a director or an audience to the audience/director table and user1 table impossible with sequential statements. Hence, one should use START TRANSACTION and COMMIT statements to avoid this issue. I would have avoided from this complexity with implementing the ISA hierarchy with just two entites of director and audience since we have the covering constraint. But then I thought; in most of the time, I would only need the username information of these entites (since it is the primary key) when I use the database and make some calculations in the sub-queires. So, I wanted to make these tables as minimal as possible with storing common attributes in the parent entity of user1.

2) When I name some entities/relations/attributes I added 1 as postfix, if they are keywords of the mysql without adding that postfix.

PROJECT-3

1) This constraint still satisfied with the NOT NULL keyword.

2) This constraint satisfied at the backend code. Whenever a session is created in the UI, our code **fills the slots with the duration attribute** of the given movie. (e.g. Let's say a director wants to add a movie to the some theater in some date. If he writes the slot as 2 and the duration is 2, we automatically fill the slot 2 as well as the slot 3. With this logic, if someone else tries to fill that slot after this operation, database would raise an error, and that error would show up in the UI.) So no movie sessions can overlap.

3) This constraint still satisfied in the SQL code with the CHECK keyword. *(CONSTRAINT CHK_SLOT CHECK (slot_number >= 1 AND slot_number <= 4)*

4) First I misinterpreted the given data at the Project 1 , no recursion is necessary with given data. We handled this constraint at the backend code. Whenever an audience wants to buy a ticket, we query the predecessor movies of the given movie_id. Then we query the bought_tickets of the audience. Finally, we check whether predecessor list is a subset of bought tickets. Thus, this constraint is satisfied.

5) This constraint is satisfied in the SQL code with dividing into two TRIGGERS: *ratings_bought_ticket_trigger and ratings_platform_match_trigger.* The logic of the triggers are straightforward.

6) This constraint still satisfied in the SQL code with the CHECK. *(CHECK (manager_number >= 1 AND manager_number <= 4))*

IMPORTANT NOTES:

1) We decided to change our approach to this constraint. We implemented this constraint in the backend & SQL code. In the SQL code, we added two triggers to prevent, *if an username exists in audience it can not exist in the director (Also reverse)*. The triggers are: *check_two_roles_constraint_trigger_director, check_two_roles_constraint_trigger_audience.* At the backend, whenever we add a new row to user1, we always add a row to either audience or director tables. Thus covering constarint is satisfied.

2) When we name some entities/relations/attributes we added 1 as postfix, if they are keywords of the PostgreSQL without adding that postfix. (This still exists to prevent ambiguity caused by keywords.)

Additional constraints that are stated in the requirements of the Project 3:

**1) Requirement 17 (buying ticket) constraints:** This requirement has three constraints. All of these are handled at the backend code as follows:
**(a):** There is no mechanism that restricts this, so automatically handled.
**(b):** Whenever an audience wants to buy a ticket, we query the predecessor movies of the movie_id that is associated with the given session_id. Then we query the bought tickets of the audience. Finally, we check whether predecessor list is a subset of bought tickets. Thus, this constraint is satisfied.
**(c):** Whenever an audience wants to buy a ticket, we query the total number of bought tickets for given session_id. We then query the capacity of the theater that is associated with the given session_id. Then we check whether there is a available capacity. Thus, the constraint is satisfied.

**2) Requirement 19 *(When audience rates a session, the average rating of the movie should be updated automatically in the ratings table.)***
This constraint is satisfied in the SQL code as requested. The name of the triggers are *update_movie_rating_trigger* and *update_movie_rating_trigger_2*. The logic is straightforward: Whenver a new ratings row is inserted to or updated in the ratings table, database automatically update the overall_ratings column of the associated movie_id with recalculating the average rating.

PREPARED BY: VOLKAN OZTURK 2019400033 & ARDA ARSLAN 2020400078