CS-336 Project for Summer 2015

Silvia Carbajal
Rutgers University
Email: suecarbajal@gmail.com

Yue Sun
Rutgers University
Email: sunyue7782@gmail.com

Margalit Kirsner
Rutgers University
Email: margo5492@gmail.com

***Abstract*— We introduce the CS336 project, a movie theater system that will efficiently get user's queries. Using four different type of users, each having their own user rights, with entities and attributes creates this DBMS. So far The problems that have come up , we have come with a solution. There are 8 scenarios with our DBMS that our group came up with and it should work as stated.**

I. Project Description

Cinema Chain System

A. Stage1 - The Requirement Gathering Stage.

Getting a realistic project idea that includes potential real world scenarios, with a description of the different user types along with their interactions with the system as well as the system feedback to them, according to their information needs. This stage also requires the specification of the different constraints and restrictions that need to be enforced depending on the different types of user (system interactions). The deliverables for this stage include the following items:

- A general description (in plain English) of the system (understandable by computer illiterate users).
- A specific description of 4 types of users (grouped by their data access/update rights).
- A description of detailed real world scenarios (at least 2 scenarios) representing those typical interactions between the different user types and the system (including inputs and outputs and data types).

Please insert your deliverables for Stage1 as follows:

- The general system description: A system to track the activity for a movie theater chain. The system contains the movies the chain's theaters are showing. Each movie has a title, director, genre, ticket price, location, time, and viewer rating. Moviegoers are able to create, edit, or delete an account and log in and out of that account. Within their account, moviegoers can rate movies, buy tickets for a movie showing, and see which movies they have already watched or have tickets for. Employees of the chain are able to create, delete, and edit movie listings. They can also access moviegoer information, see how many tickets are being sold for each movie, and see the average rating for each movie.
- The 4 types of users (grouped by their data access/update rights): theatre manager, ticket selling department, theatre manager and audience/moviegoer/client
  - The Moviegoer's access rights: Can create, delete, and edit their own account information. Can log in and out of their account. When logged in, they can view showings for any theater location, purchase tickets, and rate movies they have seen. Can also see a list of movies they have seen and/or rated, with associated information.
  - The Ticket Seller's access rights: Can purchase or refund tickets for any moviegoer within one theater location, modify the tickets. Can see total ticket sales for any movie or showing.
  - The Theater manager's access rights: Can create, delete or edit the movie showings for one theater location. Can also view the ticket sales and ratings for any movie shown at that location. theater manager also has a database of workers and can remove and add them.
  - The Chain manager's access rights: Can create, delete , or edit the movie information and movies for all theater locations. Can also view ticket sales and ratings for any movies shown at all locations. Can create, edit, or remove moviegoer's accounts and the information in them. The chain manager can also see the list of employees in each theater and has list of theaters. The chain manager has highest priority.
  - The real world scenarios: Please insert the real world scenarios in here, as follows.
    - Scenario 1 moviegoer/audience description: selects movie and time The audience/moviegoer user who has an account in the system logs in, scrolls through the movies that are showing and picks the movie he/she want to see, they pick a cinema that is closest to them, pick the number of tickets, after moves on to submit payment for the tickets. The ticket sale is counted for as soon as the ticket is bought. An email is sent to the user's email account with their confirmation order letter confirming that the payment went through.
    - System Data Input for Scenario1: userid, user-password, movie title, cinemaID ,number of tickets, time
    - Input Data Types for Scenario1: int, varchar, varchar, int, int, int
    - System Data Output for Scenario1: userid, movieid, cinemahallid, time
    - Output Data Types for Scenario1: int, int, int, int
    - Scenario 2 moviegoer/audience description: reg-

istering The audience/client goes onto our system,that user who is a first time user, registers, fills their desired user name and password , full name and email to where they will get their tickets and all that information about the user is stored in the system.

* System Data Input for Scenario 2: userPassword, UserName, emailAddress, first name , last name
* Input Data Types for Scenario 2: varchar , varchar, varchar, varchar, varchar
* System Data Output for Scenario 2: user ID, name, last name,
* Output Data Types for Scenario 2: int,varchar ,varchar

---

* Scenario 1 theater manager description: adding employees to database The theater manager needs 10 new employees as ushers so he hires them. the theatre manager adds each employee's information to the database. each employee has to meet the requirements the manager puts for them to get the job. the output would be the name, last name and employee ID of each employee as it is added into the system
* System Data Input for Scenario 1: adminUser, adminPassword, employee name, employee info, birthdate
* Input Data Types for Scenario 1: varchar, varchar, varchar, varchar, int
* System Data Output for Scenario 1: employee ID, first name, last name
* Output Data Types for Scenario 1: int, varchar, varchar
* Scenario 2 theater manager description: remove show time of a movie A theatre manager has authorization to remove showtimes of a movie that is not doing too well. the theatre manager checks to see which movie is not getting much ticket sales and lessens the amount of show times of that selected movie. the administrator looks for the movie in the database by title and removes one of the showtimes of the movie that is found.
* System Data Input for Scenario 2: adminUser, adminPassword, movie title, old time, new time
* Input Data Types for Scenario 2: varchar, varchar, varchar, int, int
* System Data Output for Scenario 2: movieid, new time, movie name
* Output Data Types for Scenario 2: int, int , varchar

---

* Scenario1 ticket seller description: A ticket seller is approached by a moviegoer who accidentally purchased a ticket for the wrong movie showing. The ticket seller can access the moviegoer's account, delete the accidentally purchased ticket, and instead add a ticket for the correct showing. The ticket seller would do this by looking up the customer's account by their customer ID, subtracting one ticket from the wrong showing, and adding one to the correct showing. This would also change the total number of tickets sold for both showings.
* System Data Input for Scenario1: customerID, movie showing IDs, ticket for first showing-1, ticket for second showing+1
* Input Data Types for Scenario1: int, int, int, int
* System Data Output for Scenario1: ticket for first showing-1, ticket for second showing+1, customerID
* Output Data Types for Scenario1: int, int, int
* Scenario 2 ticket seller description: A ticket seller wants to see if a particular movie is selling better at one time than another, and if any are close to selling out. The seller searches by movie title and looks up the total ticket sales for all showings of that movie. She can see a list of how many tickets are sold for each showing, and thus know which showings are most popular and if any have reached capacity. This information is also useful so that the ticket seller can advice moviegoers on which showings are least full and most likely to still have good seats remaining.
* System Data Input for Scenario1: movie title
* Input Data Types for Scenario1: varchar
* System Data Output for Scenario1: movie title, number of tickets sold
* Output Data Types for Scenario1: varchar, int

---

* Scenario 1 chain manager description: A chain manager has decided to take a failing movie out of the theater early. She needs to cancel all future showings. She first subtracts all advance ticket purchases for future showings, and then removes all future showings for that movie. The movie is still listed in the database, and information about it still exists, but there are no more showings. The chain manager still has the ability to add new showings in the future if she changes her mind.
* System Data Input for Scenario1: movie title
* Input Data Types for Scenario1: varchar
* System Data Output for Scenario1: empty list of movie showings
* Output Data Types for Scenario1: int
* Scenario 2 chain manager description: The chain manager wants to know if a movie showing in one theater is worth showing in other branches, too. She wants to see how high that movie's ticket sales and ratings are, in order to inform her decision. She looks up the movie and looks at how many

tickets it has sold alltogether, so she knows how popular it is, and at how high the rating is, so she knows if people like it or not. With this information, she can judge how likely the film is to succeed elsewhere

* System Data Input for Scenario1: movie title
* Input Data Types for Scenario1: varchar
* System Data Output for Scenario1: movie title, number of tickets sold, average rating
* Output Data Types for Scenario1: varchar, int, int

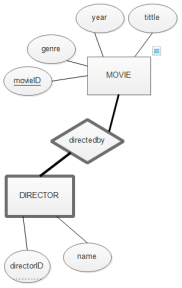
B. Stage2 - The Design Stage.

Transforming the project requirements into an ER model design elements, starting with the layout of the different data types required per user into the ER model. The deliverables for this stage include the following items, containing a graphical representation and a textual description (Please refer to the original Project Description for more details):

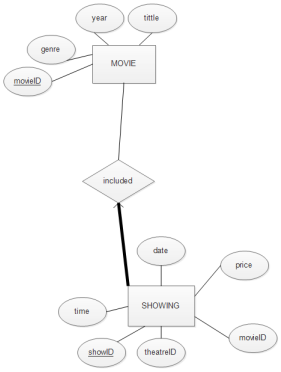
- A specification of each relation and its entities and attributes (with English description relating each ER part to one or more user scenario(s)).
- An ER diagram in its entirety (covering all the scenarios described in stage1, along with the specified integrity constraints).
- A brief textual description of the ER diagram (along with its functional operation in the different user scenarios described in the first stage of the project).
- A detailed description and justification of the integrity constraints defined in the ER.

Please insert your deliverables for Stage2 as follows:

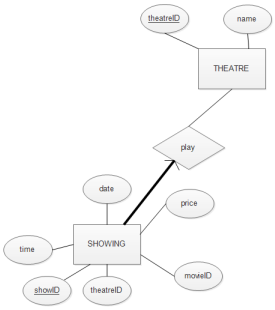
- The relation along with the attributes look like:



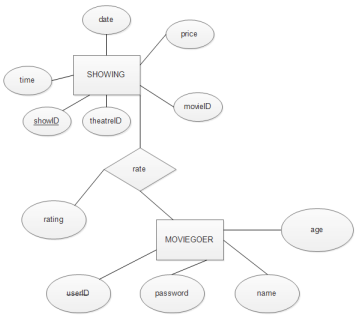
- Explanation related to user scenario: movie directed by director relation , weak entity. when movie gets deleted we take everything that comes with that movie, director. For the first moviegoer scenario, selecting a movie. The movie relation is used in the diagram, it has the attributes that a user would need. The movie table would have movieID, genre, year, and title and director has name and director ID when selecting a movie you can also select by director



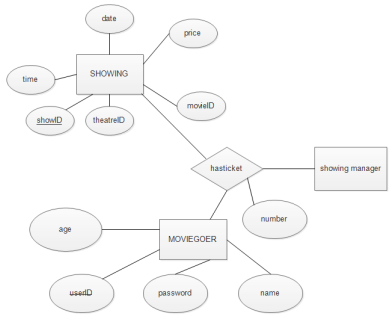
- explanation related to user scenario: showing of one movie. For the ticket seller scenario, he/she is approached by a moviegoer who purchased the wrong ticket for the wrong showing. Showing has attributes that would enable this transaction.



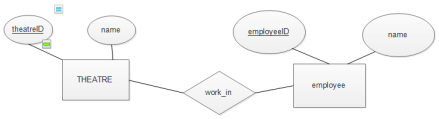
- explanation related to user scenario: For the showing manager scenario , it states that the theater manager has the ability to remove a showtime of a movie. this relation shows theater attributes and showing attributes that can allow the showing manager to remove showtimes



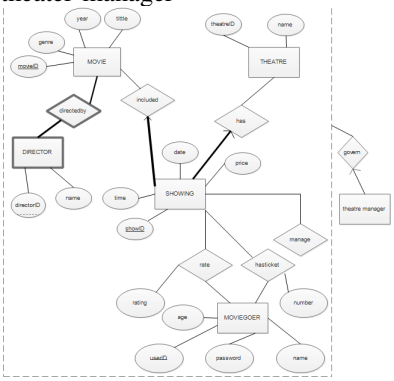
- explanation related to user scenario: moviegoer can rate showing , each of their attributes allow a user to rate a showing by movie ID



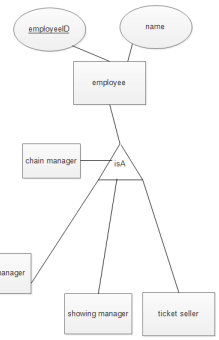
- explanation related to user scenario: This ternary relationship is between ticket seller , moviegoer and showing . For the ticket seller scenario, he/she is approached by a moviegoer who purchased the wrong ticket for the wrong showing.The ticket seller has access to the user accounts, and it is depicted in the relationship has ticket between showing and moviegoer because the tickets seller has all tickets of the moviegoer and is able to change it.



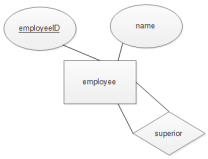
- explanation related to user scenario: many employees work in each theater ,relationship called employeeWorkin, the most important employees being ticket seller, chain manager, showing manager, and theater manager



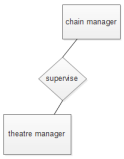
- explanation related to user scenario: a theatre manager of that theatre can manage all the stuff in that theatre, including showing, movie, and moviegoer.



- explanation related to user scenario: ticket seller, chain manager, showing manager, and theater manager are different types of employee. Each of them has different authority

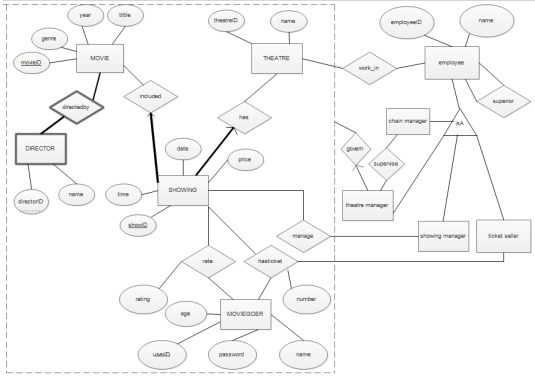


explanation related to user scenario: each employee may be superior to other employees. It's a unary relationship.



explanation related to user scenario: Chain manager supervise all the theatre manager in that chain.

- The ER diagram in its entirety:



- The ER diagram description (corresponding to the user scenarios): For the moviegoer scenario the moviegoer logs into his/her account and selects a movie and buys ticket(s). The relationship between showing and moviegoer depicts this relationship. The attributes in showing are needed to buy a ticket. The attributes for showing are time, showID, theaterID, movieID, price, and date and each attribute is used in this transaction.For selecting a movie the movie relation is used in the diagram, it has the attributes that a user would need. The movie table would have movieID, genre, year, and title.The relationship between moviegoer and showing displays that the moviegoer has a ticket for the chosen showing the moviegoer picked. The moviegoer's attributes are what we need to make this happen , which leads into the second user scenario. For the moviegoer to register in our system we would need the userID, password, and name. To buy a ticket a user would have to register in the system therefore these attributes for moviegoer are important.
- For the chain manager scenario , the chain manager adds employees to the database. The relationship between employee and theater depicts this scenario. The employee relation has an attribute ,employeeID. which is essential to this scenario because the theater has lots of employees working for them but every person working for the theaters are employees, that is why we use the is A relationship between them all , if not a theater manager , ticket seller, or administrator then its just a regular employee which is the other employees working at the theatre. For the second scenario , it states that the showing manager has the ability to remove a showtime of a movie, the manage relationship between showing and theater manager, the attributes for the showing relation has time and date and it can be changed by the theater manager.
- For the ticket seller scenario, he/she is approached by a moviegoer who purchased the wrong ticket for the wrong showing.The ticket seller has access to the user accounts, and it is depicted in the relationship has ticket between showing and moviegoer because the tickets seller has all tickets of the moviegoer and is able to change it. For the second scenario, it states that the ticket seller wants to see if a particular movie is doing well. The ticket seller is able to do this because he/she has access , by the has tickets relationship, to the showing table, where he/she can look up a movie and its ticket sales
- For the theater manager scenario, the first scenario states that the theater manager taking out all showings of a certain movie. For this the theater manager has needs and has access to ticket sales, and showings. The theater manager is the administrator and has access to everything, relations such as movie, director, showing, theater,and moviegoer and employees because of the relationship between theater and employees. Theres an aggregation in the diagram to depict this administrator manage relationship. The second scenario for the chain manager/administrator is he/she wants to see if a movie showing in one theater is worth showing in other branches. Again the administrator has access to information of all cinemas. in the showing relation the admin has access to rating because of the rate relationship between moviegoer and showing which has an attribute of rating. Therefore the administrator can find out the ratings of any showing as well as ticket sales because of the has ticket relationship and its number attribute which is between showing and moviegoer.
- The integrity constraints defined in the ER diagram: Please insert the integrity constraints in here:
  - Integrity Constraint: Each showing must be of exactly one movie.
  - The description and justification of the integrity constraint: A showing, by definition, is a specific movie being shown in a specific place at a specific time. By this definition, there has to be a movie to be shown. There cannot be more than one movie because an audience does not watch multiple movies at once.
  - Integrity Constraint: Each showing has exactly one theater.
  - The description and justification of the integrity constraint: A showing, by definition, is a specific movie being shown in a specific place at a specific time. The theater is that specific place, and so there can only be one. And there has to be one, because there needs to be somewhere for the movie to be shown.
  - Integrity Constraint: Each movie has at least one director.
  - The description and justification of the integrity constraint: Every movie that exists was directed by someone. However, it is possible for multiple people to co-direct.
  - Integrity Constraint: Each director has at least one movie.
  - The description and justification of the integrity constraint: By definition, to be a director that person must have directed something. Furthermore, there is no reason to store a director unless they belong to a movie the theater is storing information on. It

is possible for a director to have directed multiple movies.
  - Integrity Constraint: movieID is the primary key for a movie.
  - The description and justification of the integrity constraint: All entities need a key, and no movie trait is guaranteed to be unique (two movies can share a title, for example), so a unique ID number needs to be created for each movie.
  - Integrity Constraint: showId is the primary key for a showing.
  - The description and justification of the integrity constraint: All entities need a key, and no showing trait is guaranteed to be unique- one theater could have two showings of the same movie at the same time. For this reason, a unique ID number needs to be created for each showing.
  - Integrity Constraint: userID is the primary key for a moviegoer.
  - The description and justification of the integrity constraint: All entities need a key, and no moviegoer trait is guaranteed to be uniquie (two moviegoers can have the same name, for example), so a unique ID number needs to be created for each moviegoer.
  - Integrity Constraint: theaterID is the primary key for a theater.
  - The description and justification of the integrity constraint: All entities need a key. Since there is no other information stored for a theater that is uniquely identifying, so an ID number must be created for that purpose.
  - Integrity Constraint: employeeID is the primary key for an employee.
  - The description and justification of the integrity constraint: Just like with moviegoers, it is possible for multiple employees to share a name or other attributes, so a unique ID number is necessary for each employee
  - Integrity Constraint: Each theater manager can manage at most one theater.
  - The description and justification of the integrity constraint: A manager of multiple theaters is a different job (admin). A manager typically manages exactly one theater, but it is possible for a manager to not yet be assigned a theater, or to be a substitute and not belong to a specific theater.

C. Stage3 - The Implementation Stage.

Building the corresponding relational tables, according to the proposed ER model described in the previous phase enforcing the different integrity constraints. The deliverables for this stage include the following items (Please refer to the original Project Description for more details):

- The SQL tables that represent the ER project model, along with at least 3-5 rows of concrete data per table.
- The normalization steps for each table, along with explanations/justifications of each normalization step.
- The SQL table after the normalization steps (showing all table attributes).
- The SQL statements used to create the SQL tables, including the required triggers as well as the integrity constraints. At least 2 triggers and 2 of each of the following constraint types have to exist in the project tables overall:
  - Data-range constraints for certain fields (for example: the age field should be between 10 and 99, while all other entries should be rejected by the system).
  - Whether some users will be denied access and/or updates to some data according to their roles (for example: student1 can not access other students' ' grades, so a violation error pops up upon that action. Another example: a sales person can see an item price, but can not change it, since only a manger can, also a violation error pops up upon that update attempt).
- So each table should be followed by the following items:
  - Explanation for each constraint, providing examples of certain data access and/or update restrictions per user type.
  - Explanation/justification for each trigger, providing examples of certain data access and/or updates per user type.

Please insert your deliverables for Stage3 as follows:

- The SQL Table, including data entries:

moviegoer table

userID	name	password	age
ssc100	silvia	lulu24	21
margo100	margo	margo22	22
yue200	yue	yue23	23

movie table

movie_ID	year	title	genre
1	2008	The Strangers	horror
2	2015	Jurrasic World	science fiction
3	2015	Minions	family comedy
4	1999	The Blair Witch Project	horror

director table

D_id	name	movieID
1	Pierre Coffin	3
2	Kyle Balda	3
3	Colin Trevorrow	2
4	Bryan Bertino	1
5	Daniel Myrick	4
6	Eduardo Sanchez	4

Theatre

theatreID	name
1	AMC
2	BMC
3	CMC

rate

rating	movieID	userID
9	3	yue200
8	2	margo100
7	1	ssc100

Employee

employeeID	name	type
77	Jack	cm
88	ted	tm
99	Paul	sm
100	Roy	ts

```
mysql> select * from employeeWorkin
    -> ;
+------------+-----------+
| employeeID | theatreID |
+------------+-----------+
|         77 |         1 |
|         88 |         1 |
|         99 |         1 |
+------------+-----------+
```

```
mysql> select * from Theatremanagedby
    -> ;
+-----------+------------------+
| theatreID | theatremanagerID |
+-----------+------------------+
|       124 |               77 |
|       123 |               76 |
|       123 |               75 |
+-----------+------------------+
```

```
mysql> select * from Showingmanagedby
    -> ;
+-----------+------------------+
| showingID | showingmanagerID |
+-----------+------------------+
|       221 |               99 |
|       221 |              101 |
|       222 |              111 |
+-----------+------------------+
```

```
mysql> select * from superior
    -> ;
+------------+--------------+
| employeeID | supervisorID |
+------------+--------------+
|         88 |           77 |
|         99 |           88 |
|        100 |           99 |
+------------+--------------+
```

SHowing table

showID	sTime	sDate	price	movieID	theaterID
1	1930	8272015	10	1	101
2	2130	6112015	10	2	2
3	2200	6252015	9	3	3

HasTicket table

number	showID	userID
1	1	1
2	2	2
3	3	3

```
mysql> describe Rate
    -> ;
```

Field	Type	Null	Key	Default	Extra
rating	int(11)	NO		NULL	
movieID	int(11)	NO	PRI	0	
userID	varchar(40)	NO	PRI		

```
mysql> select * from supervise;
```

chainmanagerID	theatremanagerID
77	88
77	32
77	38

```
mysql> select * from govern;
```

theatremanagerID	theatreID	showingID	movieID
88	1	1	1
88	1	1	2
88	1	1	3

- The normalization steps for each table, along with explanations/justifications: The normalization steps should be inserted as follows.
  - The normalization step for the SQL table: Theater Normalization already in 3NF
  - The explanations/justification of the normalization step: It passes 1NF because each attribute can only have one value in a given row. Each theater has one ID and one name. It passes 2NF and 3NF because the only non-key attribute, name, attribute follows from the theaterID, which is the primary key.
  - The normalization step for the SQL table: Showing Normalization already in 3NF
  - The explanations/justification of the normalization step: It passes 1NF because each attribute can only have one value in a given row. Each showing has only one ID, place, date, time, and movie being shown. It passes 2NF and 3NF because the non-key attributes depend on the entire primary key alone. The place, date, time, and movie are all independent from each other, and only follow from the showID.
  - The normalization step for the SQL table: HasTicket Normalization already in 3NF
  - The explanations/justification of the normalization step: It passes 1NF because each attribute can only have one value in a given row. Each row consists of only one moviegoer with one amount of tickets for one showing. It passes 2NF and 3NF because the non-key attribute, number of tickets, depends on the entire primary key alone. The number of tickets relies on being for a particular moviegoer to use at a particular showing- in other words, both parts of the primary key.
  - The normalization step for the SQL table: Rate Normalization already in 3NF
  - The explanations/justification of the normalization step: It passes 1NF because each attribute can only have one value in a given row. Each rate consists of one rating from one moviegoer for one movie. It passes 2NF and 3NF because the non-key attribute, rating, depend on the entire primary key alone. The rating is from a specific moviegoer for a specific movie, so it relies on both parts of the primary key.
  - The normalization step for the SQL table: moviegoer Normalization already in 3NF
  - The explanations/justification of the normalization step: it passes 1NF because each attribute can only have one value in each row. Each moviegoer has a unique userID. it passes 2NF and 3NF because the non key attributes depend on the entire primary key alone. super key being movie ID . year, title and genre are all independent from each other
  - The normalization step for the SQL table: movie Normalization already in 3NF
  - The explanations/justification of the normalization step: it passes 1NF because each attribute can only have one value in each row. Each movie has a unique movie ID. . it passes 2NF and 3NF because the non key attributes depend on the entire primary key alone. primary key being user ID . name, password, and age are all independent from each other
  - The normalization step for the SQL table: director Normalization already in 3NF
  - The explanations/justification of the normalization step: it passes 1NF because each attribute can only have one value in each row. Each director has a unique director ID. It passes 2NF and 3NF because the non-key attribute, rating, depend on the entire primary key alone. There can be many directors for a movie so the director is from a specific movie and has a unique Director ID
  - The normalization step for the SQL table: Employee Normalization already in 3NF
  - The explanations/justification of the normalization step: It passes 1NF because each attribute can only have one value in a given row. Each employee has one ID one type and one name. It passes 2NF and 3NF because the non-key attributes, name and type, attribute follows from the EmployeeID, which is the primary key
  - The normalization step for the SQL table: employeeWorkin Normalization already in 3NF
  - The explanations/justification of the normalization step: It passes 1NF because each attribute can only have one value in a given row. Each hasEmployee has only one theatreID and employeeID being shown. It passes 2NF and 3NF because there is no non-key attributes in the table.
  - The normalization step for the SQL table: Theatremanagedby Normalization already in 3NF
  - The explanations/justification of the normalization step: It passes 1NF because each attribute can only have one value in a given row. Each Theatremanagedby has only one theatreID and theatremanagerID being shown. It passes 2NF and 3NF because there is no non-key attributes in the table.
  - The normalization step for the SQL table: Showingmanagedby Normalization already in 3NF
  - The explanations/justification of the normalization step: It passes 1NF because each attribute can only have one value in a given row. Each Showingmanagedby has only one showingID and showingmanagerID being shown. It passes 2NF and 3NF because there is no non-key attributes in the table
  - The normalization step for the SQL table: superior Normalization already in 3NF
  - The explanations/justification of the normalization step: It passes 1NF because each attribute can only have one value in a given row. Each superior has only

one employeeID and supervisorID being shown. It passes 2NF and 3NF because the non-key attributes, supervisorID, attribute follows from the employeeID, which is the primary key

- The SQL statement to create the table:

```
CREATE TABLE Moviegoer(
    -> userID varchar(40) NOT NULL,
    -> name varchar(100) NOT NULL,
    -> password varchar(30) NOT NULL,
    -> age int NOT NULL,
    -> PRIMARY KEY (userID),
    -> CHECK (age > 1),
    -> CHECK (age < 100));
```

- The Triggers used in the statement: Please insert the triggers are as follows:

```
create trigger usercount
after insert on Moviegoer
for each row
set @usercount=@usercount+1;
```

  - The Trigger:
  - The Trigger Explanation/Justification: a count that is added 1 after every insert on the moviegoer table to count how many moviegoers are in the system
  - The Trigger Examples (in correlation with the user(s) interaction with the system): the theater manager wants to see how many moviegoers are in his theater ,he can just summon this trigger customer count

- The Data Range Constraints used in the statement: Please insert the Data Range Constraints as follows:
  - The Data Range Constraint: Age in the moviegoer table must be between 1 and 100
  - The Data Range Constraint Explanation: ages 1 to 100 are common for a typical person
  - The Data Range Constraint Example(s) (in correlation with the user(s) interaction with the system): A person who is 25, puts their age as 25 and is accepted and valid another example is if the user puts in 255 but the last 5 is put in by mistake, the system will not take age 255 so the insert is invalid
  - The Data Access/update Constraint:

```
create view forMoviegoer(title,genre,year,sTime,sDate,price,directorname,rating)
        as select title,genre,year,sTime,sDate,price,name,rating
        from Movie,Showing,Director,Rate
        where Movie.movie_ID=Showing.movieID
                and Movie.movie_ID=Director.movieID
                and Movie.movie_ID=Rate.movieID;
```

  - The Data Access/update Constraint Explanation: For the moviegoer view , the moviegoer (client) cannot look at other users ID or passwords, name of employees or what type is each employee therefore, we create a view solely for the moviegoer , so they can have access to the movie titles, year, genre, date, time,directors, price and theater name so the moviegoer is able to search movies by director, by title, by date, time and a moviegoer is also able to see price so they can buy tickets
  - The Data Access/update Constraint Example(s) (in correlation with the user(s) interaction with the system): When a moviegoer goes to buy a ticket and wants to search by title.They search by title ?the strangers ? and find the times for it and price and go ahead and buy the ticket
  - The Data Access/update Constraint:

```
create view seniorMoviegoer(userID,name,age)
        as select Moviegoer.userID, Moviegoer.name, Moviegoer.age
        from Moviegoer where Moviegoer.age>50;
```

  - The Data Access/update Constraint Explanation: The senior moviegoer view is for employees , so they know who is a senior out of their customers , so they can get benefits , such as lower ticket price.
  - The Data Access/update Constraint Example(s) (in correlation with the user(s) interaction with the system): For example if its senior Day , which is when seniors get a free movie ticket to any movie , the theater can give it to them based on this view. This view lists all senior users that exist.

- The SQL statement to create the table:

```
CREATE TABLE Director(
    -> D_id int NOT NULL AUTO_INCREMENT,
    -> name varchar(100) NOT NULL,
    -> movieID int NOT NULL,
    -> PRIMARY KEY (D_id,movieID),
    -> FOREIGN KEY(movieID) REFERENCES movie(movie_ID)
    -> UPDATE ON CASCADE
    -> DELETE ON CASCADE);

CREATE TABLE Movie(
    -> genre varchar(20) NOT NULL,
    -> movie_ID int NOT NULL AUTO_INCREMENT,
    -> year int NOT NULL,
    -> title varchar(100) NOT NULL,
    -> PRIMARY KEY(movie_ID));
```

- The Triggers used in the statement: Please insert the triggers are as follows:

```
create trigger moviecount
after insert on Movie
for each row
set @moviecount=@moviecount+1;
```

  - The Trigger:
  - The Trigger Explanation/Justification: a count variable in which one is added every time a movie is added to the system so any of the managers can know how many movies are added to their movie theater system.
  - The Trigger Examples (in correlation with the user(s) interaction with the system): a theater manager wants to know how many movies his theater has

- The SQL statement to create the table:

```
CREATE TABLE Showing{
showID INT NOT NULL AUTO_INCREMENT,
sTime INT,
sDate INT,
price INT,
movieID INT NOT NULL,
theaterID INT NOT NULL,
PRIMARY KEY(showID),
FOREIGN KEY (movieID) REFERENCES Movie(movieID),
FOREIGN KEY (theaterID) REFERENCES Theater(theaterID)
}

CREATE TABLE Theater{
theaterID INT NOT NULL AUTO_INCREMENT,
name VARCHAR(200),
PRIMARY KEY(theaterID)
}

CREATE TABLE HasTicket{
number INT,
showID INT,
userID INT,
PRIMARY KEY(showID, userID),
FOREIGN KEY (showID) REFERENCES Showing(showID)
FOREIGN KEY (userID) REFERENCES Moviegoer(userID)
}

CREATE TABLE Rate{
rating INT NOT NULL,
movieID INT,
userID INT,
PRIMARY KEY(movieID, userID),
FOREIGN KEY (movieID) REFERENCES Movie(movieID),
FOREIGN KEY (userID) REFERENCES Moviegoer(userID),
CHECK (rating>0),
CHECK (rating<11)
```


- The Data Range Constraints used in the statement: Please

insert the Data Range Constraints as follows:

- The Data Range Constraint: A rating in the 'rate' table must be between one and ten.
- The Data Range Constraint Explanation: A ten point scale is a common one for movie ratings, with 10 being the highest possible rating (the movie was fantastic) and one being the lowest (the movie was terrible). A movie must be rated within the scale for the information to be useful.
- The Data Range Constraint Example(s) (in correlation with the user(s) interaction with the system): A moviegoer tries to rate a movie they love a 15, but it is rejected because it is off the rating scale. They then rate the movie 10, and it is accepted.

- The SQL statement to create the table:

```
create table govern(theatremanagerID int(20), theatreID int(20)
showingID int(20), primary key (theatremanagementID, theatreID,
showingID, movieID) foreign key (theatremanagerID) references
employee(employeeID), foreign key(theatreID) references
theatre(theatreID), foreign key(showingID) references
Showing(showingID),foreign key(movieID) references Movie(movie_ID)
```

```
create table supervise(chainmanagerID int(20),theatremanagerID int(20)
primary key(chainmanagerID,theatremanagerID )foreign
key(chainmanagerID ) references employee(employeeID)foreign
key(theatremanagerID ) references employee(employeeID));
```

```
CREATE TABLE Employee(employeeID int(20) not null auto_increment, name varchar(20), type
varchar(2), primary key(employeeID));
ALTER TABLE employee ADD CONSTRAINT typechecker check(type='cm' or type='tm' or type='sm'
or type='ts');
```

- The Data Range Constraints used in the statement: Please insert the Data Range Constraints as follows:
  - The Data Range Constraint: type must be one of the following cm, tm, sm or ts.
  - The Data Range Constraint Explanation: an employee can only be a chain manager or ticket seller or theater manager or showing manager
  - The Data Range Constraint Example(s) (in correlation with the user(s) interaction with the system): the chain manager puts tp for a sm (showing manager)by mistake , the insert is not allowed because tp doesn't exist
- The SQL statement to create the table:

```
CREATE TABLE employeeWorkin(theatreID int(20) NOT NULL auto increment, employeeID int(20) NOT NULL)
primary key(theatreID,employeeID) FOREIGN KEY (employeID, ON DELETE CASCADE, ON UPDATE
CASCADE;
```

```
CREATE TABLE Theatremanagedby(theatreID int(20) not null auto increment, theatremanagerID in(20) not
null, primary key(theatreID, theatremanagerID) FOREIGN KEY (employeeID) REFERENCES
Employee(employeeID),FOREIGN KEY (theatremanagerID) REFERENCES Employee(employeeID),UPDATE
ON CASCADE, DELETE ON CASCADE);
```

- The Data Range Constraints used in the statement: Please insert the Data Range Constraints as follows:
  - The Data Access/update Constraint:

```
create view
forTheatreManager(title,genre,year,stime,sdate,price,directorname,rating,userid,username,age,
numberofticket)
as select
title,genre,year,sTicket,sDate,price,Director.name,rating,Moviegoer.userID,Moviegoer.name,
age,number
from Movie, Showing,Director, Rate, HasTicket, Moviegoer,
where Movie.movie_ID=Showing.movieID and Movie.movie_ID = Director.movieID and
Movie.movie_ID=Rate.movieID and HasTicket.showID=Showing.showID and HasTicket.userID=
Moviegoer.userID;
```

  - The Data Access/update Constraint Explanation: For the theater manager view , the theater manager has access to everything in the database except the employees , because the chain manager takes care of that .The theatre manager can look up cannot look at other users ID or passwords, name of employees or what type is each employee therefore, we create a view solely for the theater manager , so they can have access to the movie, showings, moviegoer entities so the theater manager can make any changes they want to movies in the system, showings and can make changes to the users in the system
  - The Data Access/update Constraint Example(s) (in correlation with the user(s) interaction with the system): When a theater manager wants to erase a showtime of a movie , it has access to do that , and other people cant

- The SQL statement to create the table:

```
CREATE TABLE Showingmanagedby(ShowingID int(20) not null auto increment, employeeID in(20) not null,
primary key(ShowingID, employeeID) FOREIGN KEY (employeeID) REFERENCES
Employee(employeeID),FOREIGN KEY (ShowingID) REFERENCES Showing(ShowingID),UPDATE ON
CASCADE);
CREATE TABLE superior(employeeID int(20) not null auto increment, supervisorID in(20) not null UNIQUE,
primary key(employeeID) FOREIGN KEY (employeeID) REFERENCES Employee(employeeID),FOREIGN
KEY (supervisorID) REFERENCES Employee (employeeID),UPDATE ON CASCADE , DELETE ON
CASCADE);
```

Repeat that pattern for every table created.

D. Stage4 - User Interface.

Describing of a User Interface (UI) to the system along with the related information that will be shown on each screen (How users will query the tables and view the query results). The emphasis should be placed on the process required for a user in order to meet a particular information need, in a user-friendly manner. The deliverables for this stage include the following items (Please refer to the original Project Description for more details):

- The SQL statements used to query the data.
- The error messages that will pop-up when users access and/or updates are denied.
- The error messages corresponding to the integrity constraints violations.
- The error messages corresponding to the data range constraints violations.
- The header of the views created in order to facilitate data accesses, according to users' needs.
- Each view created must be justified. Any triggers built upon those views should be explained and justified as well. At least one view should be created and justified for the project.

Please insert your deliverables for Stage4 as follows:

- The the first SQL statement used to query the data:

```
INSERT INTO Movie(genre,year,title) VALUES(horror,1980,frankestein);
```

- The tables taking part in the query statement (table names and headers): Please insert the tables taking part in the query statement, as follows.
  - The name of the first table taking part in the query: Movie

  - The header of the table (all attributes):

```
mysql> describe Movie;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| genre    | varchar(20) | NO   |     | NULL    |                |
| movie_ID | int(11)     | NO   | PRI | NULL    | auto_increment |
| year     | int(11)     | NO   |     | NULL    |                |
| title    | varchar(100)| NO   |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
```

  - The attributes of the table taking part in the query: genre,year,title
- The error messages popping-up when users access and/or updates are denied (along with explanations and examples):
  - The error message: ERROR 1142(42000): INSERT command denied to user ?moviegoer?@?localhost? for table ?Movie?
  - The error message explanation (upon which violation does it take place): When a moviegoer tries to make changes to the movie information, this will be denied because the moviegoer does not have access to movie information
  - The error message example according to user(s) scenario(s): When a moviegoer user tries to make add a movie to the movie table because he wants to add one of his favorite movies that arent showing in any of theaters , he/she tries to add it and the error message will pop-up.
- The the first SQL statement used to query the data:

```
INSERT INTO Showing(sTime,sDate,price,movieID,theaterID)
VALUES(1930,8222015,5,3);
```

- The tables taking part in the query statement (table names and headers): Please insert the tables taking part in the query statement, as follows.
  - The name of the first table taking part in the query: Showing
  - The header of the table (all attributes):

```
mysql> describe Showing;
+-----------+---------+------+-----+---------+----------------+
| Field     | Type    | Null | Key | Default | Extra          |
+-----------+---------+------+-----+---------+----------------+
| showID    | int(11) | NO   | PRI | NULL    | auto_increment |
| sTime     | int(20) | YES  |     | NULL    |                |
| sDate     | int(20) | YES  |     | NULL    |                |
| price     | int(11) | YES  |     | NULL    |                |
| movieID   | int(11) | NO   | MUL | NULL    |                |
| theaterID | int(11) | NO   | MUL | NULL    |                |
+-----------+---------+------+-----+---------+----------------+
```

  - The attributes of the table taking part in the query: showID,sTime,sDate,price,movieID,theaterID
- The error messages popping-up when users access and/or updates are denied (along with explanations and examples):
  - The error message: ERROR 1142(42000): INSERT command denied to user ?moviegoer?@?localhost? for table ?Showing?
  - The error message explanation (upon which violation does it take place): When a moviegoer tries to make changes to the showing information, the require will be denied because the moviegoer do not have access to showing information
  - The error message example according to user(s) scenario(s): When a moviegoer user tries to make changes to showing information to mess with one of the theaters, the error message will pop-up.
- The the first SQL statement used to query the data:

```
INSERT INTO Showing(showID,sTime,sDate,price,movieID,theaterID)
VALUES(2, 1215, 9142015, 10, 2, 2);
```

- The tables taking part in the query statement (table names and headers): Please insert the tables taking part in the query statement, as follows.
  - The name of the first table taking part in the query: Showing
  - The header of the table (all attributes): Showing(showID,sTime,sDate,price,movieID,theaterID)

```
mysql> DESCRIBE Showing;
+-----------+---------+------+-----+---------+----------------+
| Field     | Type    | Null | Key | Default | Extra          |
+-----------+---------+------+-----+---------+----------------+
| showID    | int(11) | NO   | PRI | NULL    | auto_increment |
| sTime     | int(20) | YES  |     | NULL    |                |
| sDate     | int(20) | YES  |     | NULL    |                |
| price     | int(11) | YES  |     | NULL    |                |
| movieID   | int(11) | NO   | MUL | NULL    |                |
| theaterID | int(11) | NO   | MUL | NULL    |                |
+-----------+---------+------+-----+---------+----------------+
```

  - The attributes of the table taking part in the query: showID,sTime,sDate,price,movieID,theaterID

  Please repeat that pattern for all tables taking part in that query.
- The error messages corresponding to the integrity constraints violations (along with explanations and examples).
  - The error message: ”Cannot duplicate an ID; must be unique.”
  - The error message explanation (upon which violation does it take place): This error is for when a user tries to make a new movie showing with a showing ID that is already used for another showing. This violates the primary key intergrity constraint.
  - The error message example according to user(s) scenario(s): An employee tries to create a showing with an already existing ID. The error message catches this and explains that each showing must have its own ID.
- The the first SQL statement used to query the data:

```
INSERT INTO Rate(rating,movieID,userID) VALUES(7, NULL, 0);
```

- The tables taking part in the query statement (table names and headers): Please insert the tables taking part in the query statement, as follows.
  - The name of the first table taking part in the query: Rate
  - The header of the table (all attributes):

```
mysql> DESCRIBE Rate;
+---------+---------+------+-----+---------+-------+
| Field   | Type    | Null | Key | Default | Extra |
+---------+---------+------+-----+---------+-------+
| rating  | int(11) | NO   |     | NULL    |       |
| movieID | int(11) | NO   | PRI | 0       |       |
| userID  | int(11) | NO   | PRI | 0       |       |
+---------+---------+------+-----+---------+-------+
```

  - The attributes of the table taking part in the query: rating,movieID,userID

- The error messages corresponding to the integrity constraints violations (along with explanations and examples).
  - The error message: Null was entered into a field that cannot be null
  - The error message explanation (upon which violation does it take place): This error is for when a user tries to rate a movie without specifying a movie ID. This violates the integrity constraint that each rating must be for exactly one movie.
  - The error message example according to user(s) scenario(s): A user enters a rating, but does not specify the movie that the rating is for. Since the rating has to belong to a movie, that action is not allowed
- The the first SQL statement used to query the data:

```
INSERT INTO moviegoer
    VALUES ('yue200','yue','yue23','100');
```

- The tables taking part in the query statement (table names and headers): Please insert the tables taking part in the query statement, as follows.
  - The name of the first table taking part in the query: Moviegoer
  - The header of the table (all attributes):

Field	Type	Null	Key	Default	Extra
userID	varchar(40)	NO	PRI	NULL	
name	varchar(100)	NO		NULL	
password	varchar(30)	NO		NULL	
age	int(11)	NO		NULL	

  - The attributes of the table taking part in the query: userID, name, password, age
- The error messages corresponding to the data range constraints violations (along with explanation).
  - The error message: invalid age
  - The error message explanation (upon which violation does it take place): 1 to 100 is accepted because a person who normally goes to movies can be between 1 and 100
  - The error message example according to user(s) scenario(s): When a user types in their age when they are filling out their information we need, and they accidentally type 555 instead of 55 , the error message will pop up.
- The the first SQL statement used to query the data:

```
INSERT INTO Rate(rating,movieID,userID) VALUES(11, 1, 0);
```

- The tables taking part in the query statement (table names and headers): Please insert the tables taking part in the query statement, as follows.
  - The name of the first table taking part in the query: Rate
  - The header of the table (all attributes):

```
mysql> describe Rate
    -> ;
```

Field	Type	Null	Key	Default	Extra
rating	int(11)	NO		NULL	
movieID	int(11)	NO	PRI	0	
userID	varchar(40)	NO	PRI		

  - The attributes of the table taking part in the query: rating,movieID,userID
- The error messages corresponding to the data range constraints violations (along with explanation).
  - The error message: invalid rating
  - The error message explanation (upon which violation does it take place): rating is allowed from number 1 to 10 . if the moviegoer liked the movie then he/she would give it a 10 , otherwise if he/she did not like it at all they can give it a 1. rating between 1 and 10 no more or less, because it is usually one of the common scales out there, whereas if its 1 to 100 rating
  - The error message example according to user(s) scenario(s): if someone really liked a movie and wants to give them 100 rating , it will not be allowed because it would not be fair to the other movies, and it is not allowed because it is not the 1-10 common scale we are using for this theater management
- The header of the views created in order to facilitate data accesses: Please insert the header of the views as follows:
  - The view created: forMoviegoer
  - The view justification: This view is built for moviegoer. When moviegoers log in, they have access to read all the movie information.
  - The tables taking part in the view and the specific attributes taking part in the view: Please insert the tables names, headers, and the attributes as follows.
    - The table name: Movie
    - The table header (all attributes):

```
mysql> describe Movie;
```

Field	Type	Null	Key	Default	Extra
genre	varchar(20)	NO		NULL	
movie_ID	int(11)	NO	PRI	NULL	auto_increment
year	int(11)	NO		NULL	
title	varchar(100)	NO		NULL	

    - The table attributes that take part in the view: title, genre, year
    - The table name: Showing
    - The table header (all attributes):

```
mysql> describe Showing;
```

Field	Type	Null	Key	Default	Extra
showID	int(11)	NO	PRI	NULL	auto_increment
sTime	int(20)	YES		NULL	
sDate	int(20)	YES		NULL	
price	int(11)	YES		NULL	
movieID	int(11)	NO	MUL	NULL	
theaterID	int(11)	NO	MUL	NULL	

    - The table attributes that take part in the view: sTime, sDate, price

* The table name: Director
* The table header (all attributes):

```
mysql> describe Director
    -> ;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| D_id     | int(11)      | NO   | PRI | NULL    | auto_increment |
| name     | varchar(100) | NO   |     | NULL    |                |
| movieID  | int(11)      | NO   | PRI | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
```

* The table attributes that take part in the view: directorname.
* The table name: Rate
* The table header (all attributes):

```
mysql> describe Rate
    -> ;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| rating   | int(11)     | NO   |     | NULL    |       |
| movieID  | int(11)     | NO   | PRI | 0       |       |
| userID   | varchar(40) | NO   | PRI |         |       |
+----------+-------------+------+-----+---------+-------+
```

* The table attributes that take part in the view: rating

Please repeat that pattern for every table that takes part in the view.

– The SQL statement used for creating the view:

```
create view forMoviegoer(title,genre,year,sTime,sDate,price,directorname,rating)
    as select title,genre,year,sTime,sDate,price,name,rating
    from Movie,Showing,Director,Rate
    where Movie.movie_ID=Showing.movieID
        and Movie.movie_ID=Director.movieID
        and Movie.movie_ID=Rate.movieID;
```

– The trigger built upon those views (if any): Please insert the triggers as follows.

```
create trigger usercount
after insert on Moviegoer
for each row
set @usercount=@usercount+1;
```

* The trigger:
* The trigger justification (explanation, correlating it to user scenario(s)): a count that is added 1 after every insert on the moviegoer table to count how many moviegoers are in the system.the theater manager can see how many moviegoers are in his theater ?,he can just summon this trigger customer count

Please, in case you have more than one view, repeat that pattern for each view.

– The view created: forTheatreManager
– The view justification: This view is built for theatre manager. When theatre manager log in, they have access to read all the movie information, showing information and the associated moviegoer information.
– The tables taking part in the view and the specific attributes taking part in the view: Please insert the tables names, headers, and the attributes as follows.
* The table name: Movie
* The table header (all attributes):

```
mysql> describe Movie;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| genre    | varchar(20)  | NO   |     | NULL    |                |
| movie_ID | int(11)      | NO   | PRI | NULL    | auto_increment |
| year     | int(11)      | NO   |     | NULL    |                |
| title    | varchar(100) | NO   |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
```

* The table attributes that take part in the view: title, genre, year
* The table name: Showing
* The table header (all attributes):

```
mysql> describe Showing;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| showID     | int(11)  | NO   | PRI | NULL    | auto_increment |
| sTime      | int(20)  | YES  |     | NULL    |                |
| sDate      | int(20)  | YES  |     | NULL    |                |
| price      | int(11)  | YES  |     | NULL    |                |
| movieID    | int(11)  | NO   | MUL | NULL    |                |
| theaterID  | int(11)  | NO   | MUL | NULL    |                |
+------------+----------+------+-----+---------+----------------+
```

* The table attributes that take part in the view: sTime, sDate, price
* The table name: Director
* The table header (all attributes):

```
mysql> describe Director
    -> ;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| D_id     | int(11)      | NO   | PRI | NULL    | auto_increment |
| name     | varchar(100) | NO   |     | NULL    |                |
| movieID  | int(11)      | NO   | PRI | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
```

* The table attributes that take part in the view: directorname
* The table name: Moviegoer
* The table header (all attributes):

```
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| userID   | varchar(40) | NO   | PRI | NULL    |       |
| name     | varchar(100)| NO   |     | NULL    |       |
| password | varchar(30) | NO   |     | NULL    |       |
| age      | int(11)     | NO   |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
```

* The table attributes that take part in the view: username,age
* The table name: HasTicket
* The table header (all attributes):

```
mysql> describe HasTicket
    -> ;
+----------+---------+------+-----+---------+-------+
| Field    | Type    | Null | Key | Default | Extra |
+----------+---------+------+-----+---------+-------+
| number   | int(11) | YES  |     | NULL    |       |
| showID   | int(11) | NO   | PRI | 0       |       |
| userID   | int(11) | NO   | PRI | 0       |       |
+----------+---------+------+-----+---------+-------+
```

* The table attributes that take part in the view: number.

Please repeat that pattern for every table that takes part in the view.

– The SQL statement used for creating the view:

```
create view
forTheatreManager(title,genre,year,stime,sdate,price,directorname,rating,userid,username,age,
numberofticket)
as select
title,genre,year,sTicket,sDate,price,Director.name,rating,Moviegoer.userID,Moviegoer.name,
age,number
from Movie, Showing,Director, Rate, HasTicket, Moviegoer,
where Movie.movie_ID=Showing.movieID and Movie.movie_ID = Director.movieID and
Movie.movie_ID=Rate.movieID and HasTicket.showID=Showing.showID and HasTicket.userID=
Moviegoer.userID;
```

– The trigger built upon those views (if any): Please insert the triggers as follows.

```
create trigger moviecount
after insert on Movie
for each row
set @moviecount=@moviecount+1;
```

* The trigger:
* The trigger justification (explanation, correlating it to user scenario(s)): a count variable in which one is added every time a movie is added to the system so any of the managers can know how many movies are added to their movie theater system. a theater manager can check how many movies his theater has

II. Project Highlights.

- All the related source code should run on the Ilab machines, and should be included with your submission along with the final LaTeX report, as well as the Power Point Presentation.
- The presentation (7 to 8 minutes) should include the following points (The order of the slides is important):
  1) Title: Project Names (authors and affiliations)
  2) Project Goal
  3) Outline of the presentation
  4) Description
  5) A picture is essential. Interface snapshot and users interaction with it
  6) Project Stumbling Blocks
  7) Data collection, Conceptual Diagram, Integrity constraints
  8) Sample of findings
  9) Future Extensions
  10) Acknowledgements
  11) References and Resources used(libraries, languages, web resources)
  12) Demo(3 minutes)

  Please follow the example SuperFinal presentation mock up that is posted on Sakai, for more details.
- On July 11th, submission should have taken place. Also you have to be ready to do the presentation (7 to 8 minutes) as well as the demo to show the functionalities of the project (3 minutes): every group member should attend the demo (and presentation) and will have to specify the parts he/she has worked upon, in order to allow us to grade all students with fairness.
- Thank you, and best of luck!