4.1 Database Design

The data required for this project is organized and stored as tables in MYSQL database. The lists of tables in this project are:

- BOOKING : This table consists of details of the booking made by the customer like booking id, movie name, time, number of seats, etc,.

Table 4.1: Booking

```
DESC booking
```

+ Options

Field	Type	Null	Key	Default	Extra
Book_id	int(11)	NO	PRI	*NULL*	auto_increment
Name	varchar(20)	YES	MUL	*NULL*	
No_of_seats	int(11)	NO		*NULL*	
Class	varchar(15)	NO		*NULL*	
Time	time	NO		*NULL*	
Movie_id	int(11)	YES		*NULL*	
Total	int(11)	NO		*NULL*	

- FOOD: This table consists of all the food items available and their prices.

Table 4.2: Food

```
DESC food
```

+ Options

Field	Type	Null	Key	Default	Extra
Product_id	varchar(10)	NO	PRI	*NULL*	
Description	varchar(50)	NO		*NULL*	
Price	int(11)	NO		*NULL*	

FOOD_BOOKING: This table contains the details of the food booking made by a customer

Table 4.3: Food

```
DESC food_booking
```

+ Options

Field	Type	Null	Key	Default	Extra
Book_id	int(11)	NO	MUL	*NULL*	
Product_Id	varchar(10)	NO	MUL	*NULL*	
Quantity	int(11)	NO		*NULL*	
Total	int(11)	NO		*NULL*	

- LOGIN: This table contains the login details of the user like user id, username, password, etc.

Table 4.4: Login

```
DESC login
```

+ Options

Field	Type	Null	Key	Default	Extra
userid	int(100)	NO	PRI	*NULL*	auto_increment
uname	varchar(100)	NO		*NULL*	
pass	varchar(100)	NO		*NULL*	
email_id	varchar(100)	NO		*NULL*	

- MOVIE: This table consists of details of the movies currently being screened at the multiplex like movie id, movie name, and auditorium where it is screened.

Table 4.5: Movie

```
DESC movie
```

+ Options

Field	Type	Null	Key	Default	Extra
Movie_id	varchar(10)	NO	PRI	*NULL*	
name	varchar(30)	NO		*NULL*	
Audi	int(11)	NO		*NULL*	

- SEATS: This contains the details of the classes of seats available like Silver, Gold and Premium for a movie.

Table 4.5: Seats

```
DESC seats
```

+ Options

Field	Type	Null	Key	Default	Extra
Movie_id	int(11)	NO		*NULL*	
Silver	int(11)	NO		20	
Gold	int(11)	NO		20	
Premium	int(11)	NO		20	
Time	time	NO		*NULL*	

The Schema (Figure 4.1) depicts the dependencies among the tables and the Entity-Relation diagram (Figure 4.2) depicts the relations and their corresponding entities.

4.2 Database Schema

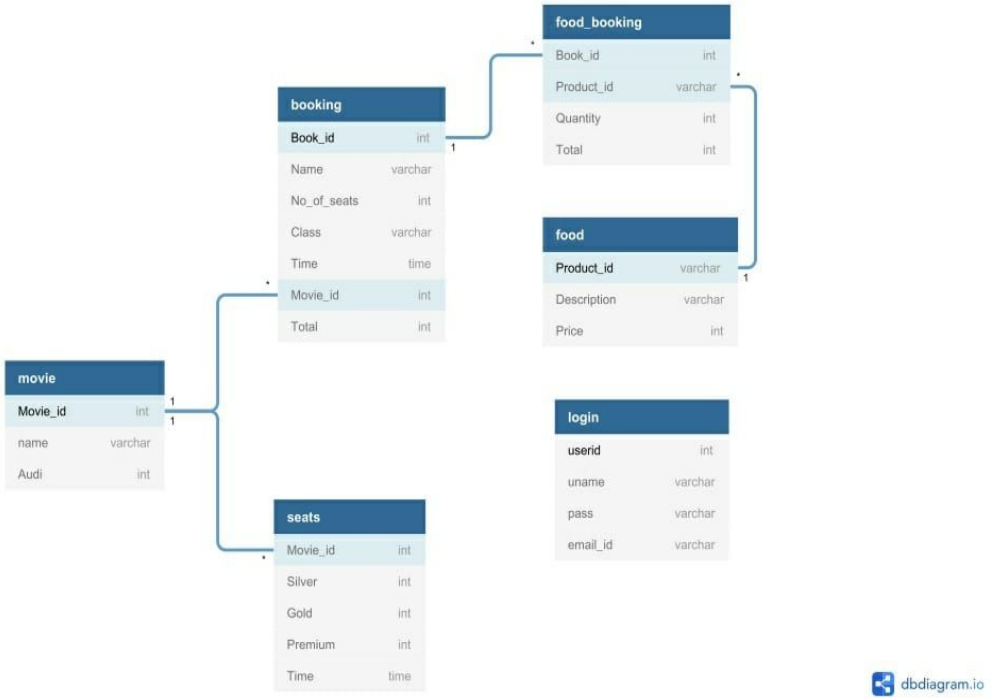


Figure 4.1: Schema Diagram

4.2 Entity Relationship Diagram

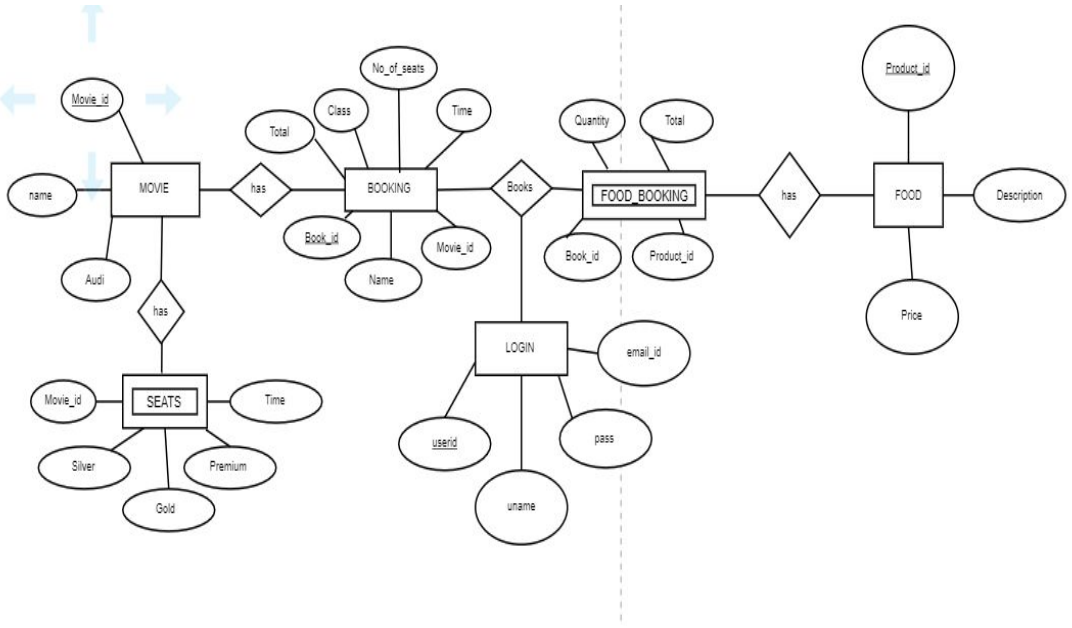


Figure 4.2: The ER Diagram

Implementation

The term implementation has different meanings, ranging from the conversion of a basic application to a compatible replacement of a computer system. Implementation is used here to make the process of converting a new or revised system into an operational one.

During the implementation stage we convert the detailed code in a programming language. If the implementation stage is not carefully planned and controlled, it can cause great chaos. Thus it can be considered to be the most crucial stage in achieving the user confidence that the new system will work efficiently.

5.1 Table Creation

5.1.1 Creation of table 'Booking':

```
CREATE TABLE booking (
    Book_id int PRIMARY KEY, Name varchar(20),
    No_of_seats int, Class varchar(11), Time time, Movie_id int, Total int,
    FOREIGN KEY(Name) references MOVIE(name) on delete cascade,
    FOREIGN KEY(Movie_id) references MOVIE(Movie_id)
    on delete cascade
);
```

5.1.2 Creation of table 'Food':

```
CREATE TABLE food (
        Product_id varchar(10) PRIMARY KEY,
        Description varchar(50), Price int
    );
```

5.1.3 Creation of table ‘Food_Booking’:

```
CREATE TABLE food_booking (
        Book_id int, Product_id varchar(20), Quantity int, Total int, FOREIGN
        KEY(Book_id) references BOOKING(Book_id) on delete cascade,
        FOREIGN KEY(Product_id) references FOOD(Product_id) on delete
        cascade
 );
```

5.1.4 Creation of table ‘Login’:

```
CREATE TABLE login (
        userid int PRIMARY KEY, uname varchar(100),
        pass varchar(100),email_id varchar(100)
);
```

5.1.5 Creation of table ‘Movie’:

```
CREATE TABLE movie (
        Movie_id int PRIMARY KEY,
        name varchar(30), Audi int
   );
```

5.1.6 Creation of table ‘Seats’:

```
CREATE TABLE seats (
        Movie_id int, Silver int, Gold int,
        Premium int, Time time, FOREIGN KEY(Movie_id) references
        MOVIE(Movie_id) on delete cascade
      );
```

5.2 Triggers

A trigger is a procedural code that is automatically executed in response to certain events on a particular table or view in a database. Triggers are mostly used for maintaining the integrity of the information in the database.

We have introduced 3 triggers in our system:

'InvalidSeats' : This trigger checks if the number of seats entered by the user are less than one or invalid number of seats are entered.

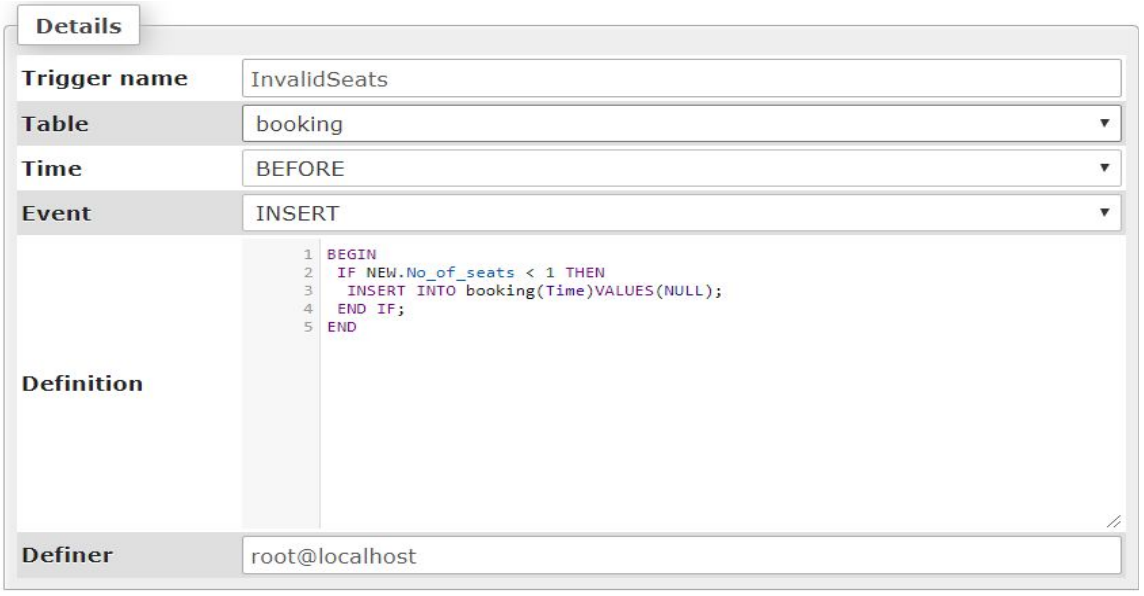


Figure 5.1: InvalidSeats

'SeatsUnavailable' : This trigger checks if the number of seats selected in a particular class are available or not.

Details

Trigger name	SeatsUnavailable
Table	booking
Time	BEFORE
Event	INSERT

Definition	BEGIN
	
	IF new.Class='Gold'
	THEN
	IF New.No_of_seats >(SELECT Gold FROM seats s WHERE New.Movie_id=s.Movie_id AND New.Time= s.Time)
	THEN
	INSERT INTO booking (No_of_seats) VALUES(NULL);
	END IF;
	END IF;
	
	IF new.Class='Silver'
	THEN
	IF new.No_of_seats >(SELECT Silver FROM seats s WHERE New.Movie_id=s.Movie_id AND New.Time= s.Time)
	THEN

Definer	root@localhost

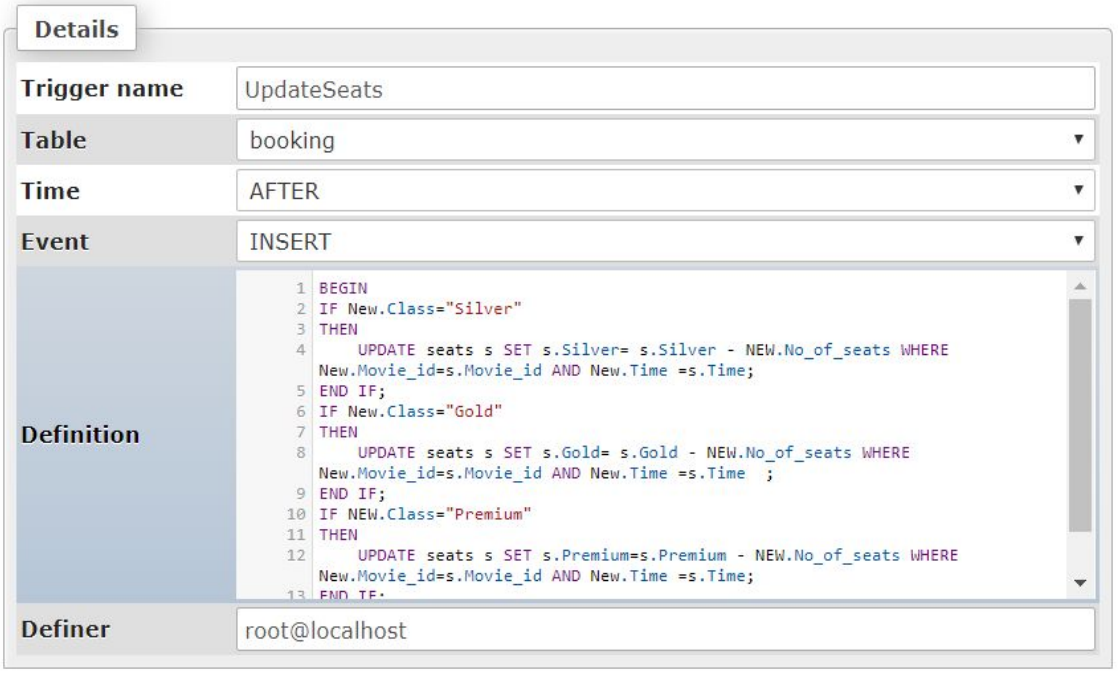
Details

Trigger name	SeatsUnavailable
Table	booking
Time	BEFORE
Event	INSERT
Definition	IF new.Class='Silver' THEN IF new.No_of_seats >(SELECT Silver FROM seats s WHERE New.Movie_id=s.Movie_id AND New.Time= s.Time) THEN INSERT INTO booking (No_of_seats) VALUES(NULL); END IF; END IF; IF new.Class='Premium' THEN IF new.No_of_seats >(SELECT Premium FROM seats s WHERE New.Movie_id=s.Movie_id AND New.Time= s.Time) THEN INSERT INTO booking (No_of_seats) VALUES(NULL); END IF;
Definer	root@localhost

Figure 5.2: SeatsUnavailable

'UpdateSeats' : This trigger is used to update the number of seats in each class after a booking has been made.

Figure 5.3: UpdateSeats



5.3 Stored Procedure

A stored procedure is a subroutine available to applications that access Relational Database Management Systems (RDBMS). Such procedures are stored in database data dictionary.

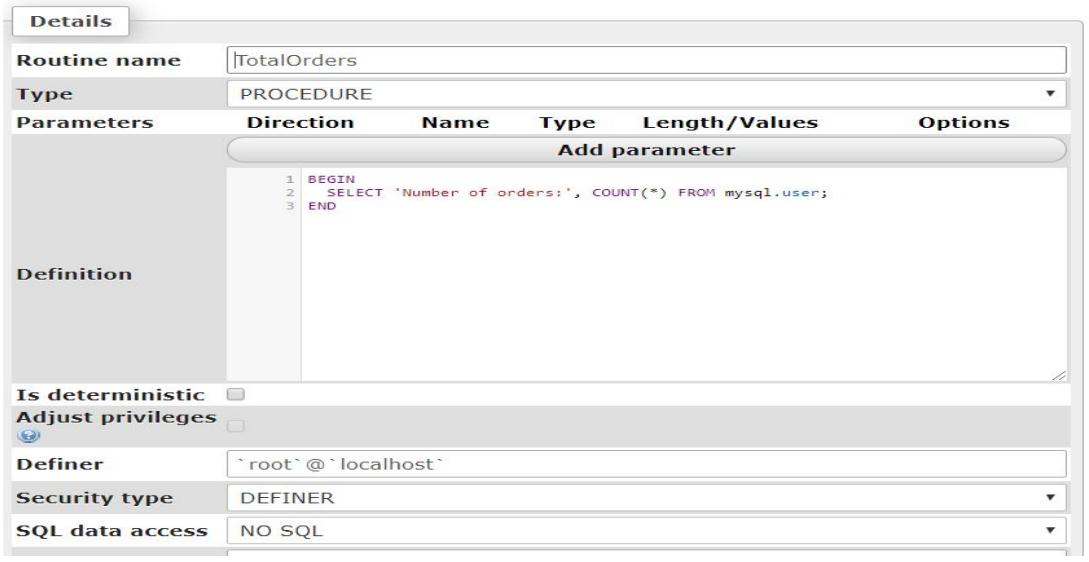


Figure 5.4: TotalOrders