DBMS PROJECT REPORT

PES UNIVERSITY

DATABASE MANAGEMENT SYSTEM

UE18CS252

SUBMITTED BY

RICHA ANGADI **PES2201800111**

TOPIC

MOVIE THEATER DATABSE MANAGEMENT SYSTEM

TABLE OF CONTENTS

INTRODUCTION

Multiplex is a movie theatre complex with multiple screens within a single complex. They are usually housed in a specially designed building.

- Movie Theatre Database System is an application of database management system which provides the user to book movie tickets and manager to keep their records.

- Movie Theatre Database system revolutionized the trivial system of distribution of Paper tickets and made it easy for users to book tickets from home.

- This system provides secure, organized and efficient storage of huge data. One does not have to be physically present at the theatre to get their jobs done.

- With the advent of various online platform like BookMyShow use type of movie theatre database system, the process became very simple, hustle free and user-friendly.

- This Project aims to provide an insight of such Movie Theatre Database System.

An entity set is a set of entities of the same type. Entity sets need not be disjoint. An entity is represented by a set of attributes.

We will be dealing with seven entity sets in this section:

- People: It is the set of all the people who book the tickets. Each Person is described by People ID, Name, Age and phone number. Attributes : p_id, p_name, age, phonenumber.

- Movie: This entity stores the details of movies to be shown on the screen in the theatre. Each movie is described by its meta-data Movie Id, movie

title, movie director, movie length and movie genre. Attributes: movie_id, movie_title, movie_director, movie_length, movie_genre.

- Booking: Booking stores data of booked tickets containing details of price, booking Id. Attributes: booking_id, price.

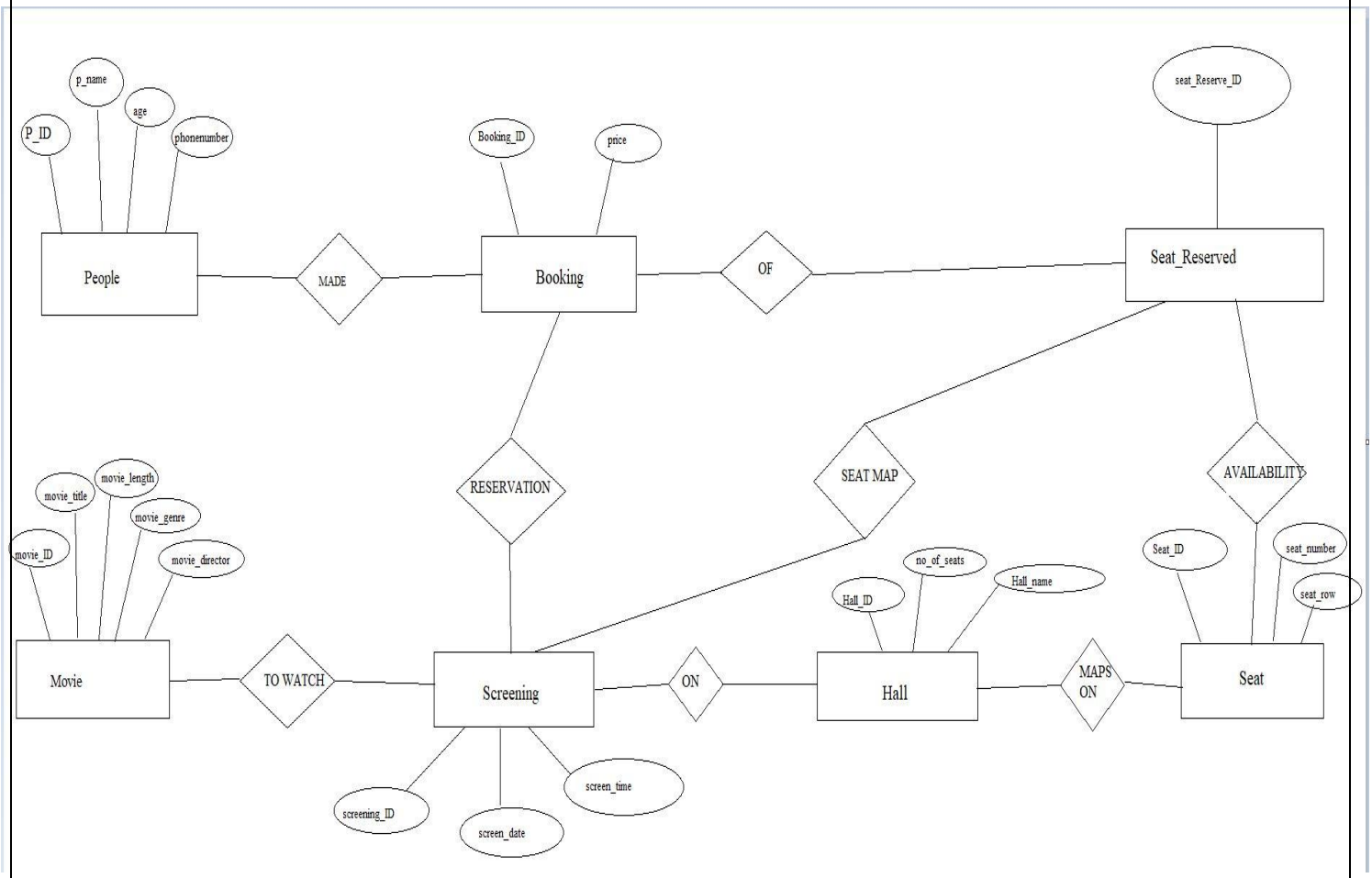
- Screening: It is the combined data of the Movie and hall in which it is shown. It describes screening id, screen time, screen data. Attributes: screening_ID, screen_time, screen_date.

- Hall: This entity stores the data of hall and seats combined described by hall id, number of seats and hall name. Attributes: hall_ID, no_of_seats, hall_name.

- Seat Reserved: Based on the booking, this entity stores the data of seat reserve id. Attributes: seat_Reserve_ID.

- Seat: This entity contains the information of seat in a hall and is described by seat id, seat row and seat number. Attributes: seat_ID, seat_row, seat_number.

DATA MODEL



We will be dealing with 8 relationship set:

• MADE: This set establishes relationship between People and Booking entities with one to many relationships between them. By this relationship using people_ID attribute (as foreign key) of booking one can find details of the person who has booked ticket.

• OF: Relationship between Booking and Seat Reserved entities is one to one relationship, as one booking can only confirm one seat. Seat Reserved entity contains booking_ID as foreign key to get details of booking.

• TO WATCH: It is the relationship between Movie and Screening entities. It has one to many relationships as one movie can be shown at different screens but a screen cannot show more than one movie at a given time. Movie_ID is kept as foreign key in screening entity.

• ON: It is the relationship between Screening and Hall and has many to one relationship as many screenings can be presented in one hall at different times and dates. Hall_ID is kept as foreign key in Screening entity.

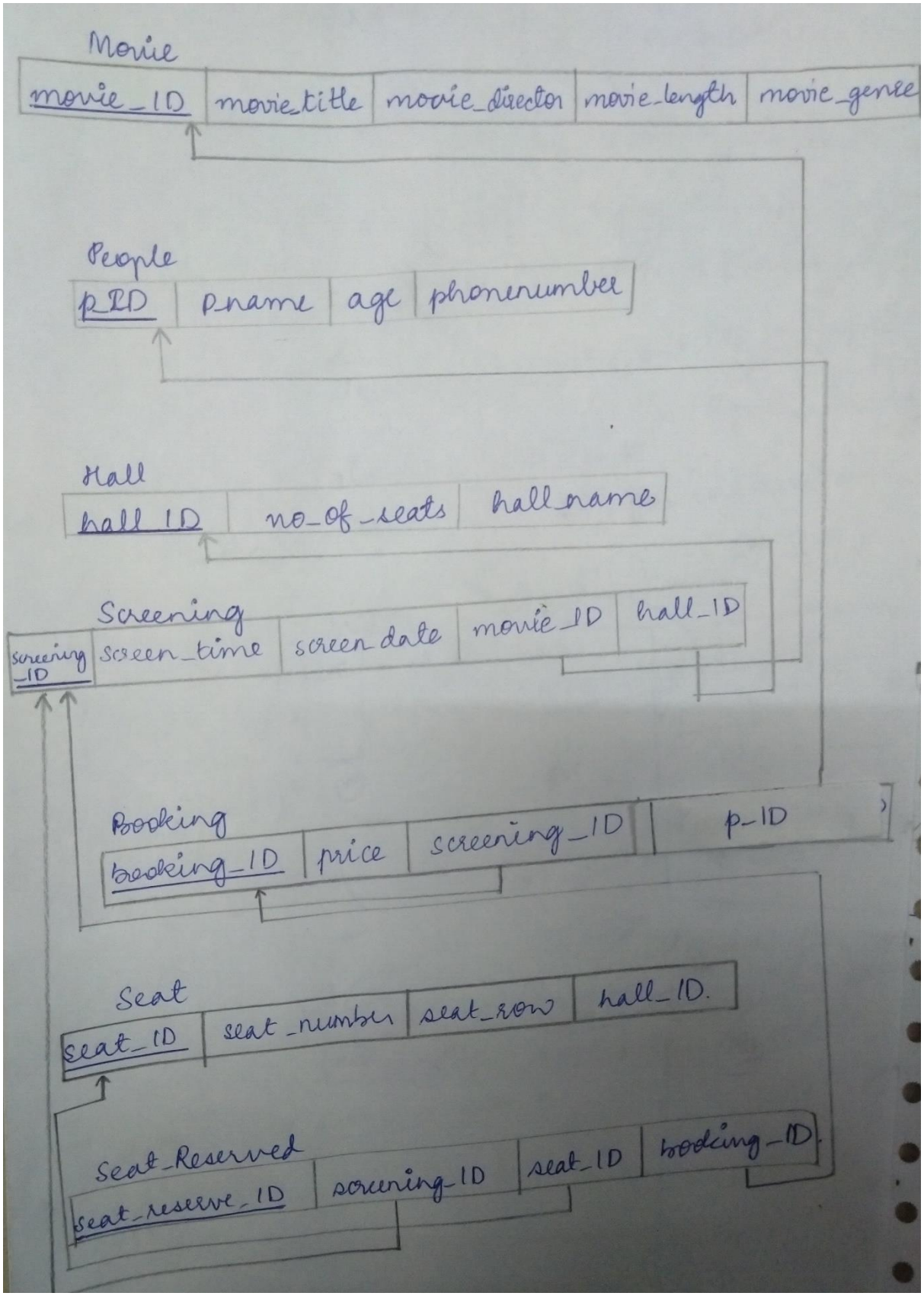
• MAPS ON: It maps relationship between Hall and Seat entity and has one to many relationships as one hall can have many seats. Seat contains Hall_ID as foreign key.

• AVAILABILITY: It is the relationship between Seat Reserved and Seat entity. It has many to one relationship as there can be multiple reservations on different times and dates of a particular seat (not more than one reservation of a seat at a given time and date). Seat Reserved entity has Seat_ID as foreign key.

• SEAT MAP: It is the relationship between Screening and Seat Reserved entities. It has one to many relationships as there are multiple seats reserved for a screening. Seat reserved entity has screening_ID as foreign Key.

• RESERVATION: It establishes relationship between Screening and Booking entities. It is one to many relationships. One screening can have multiple booking by people. Screening_ID is foreign key in Booking entity.

RELATIONAL SCHEMA



FUNCTIONAL DEPENDENCIES AND NORMALIZATION

All the functional dependencies present in this database (R is relation) are as follows:

1. p_ID -> {p_name, age, phonenumber}
2. booking_ID -> {p_ID, price, hall_ID, screening_ID}
3. seat_reserve_ID -> {booking_ID, seat_ID, hall_ID}
4. seat_ID -> {seat_number, seat_row, hall_ID}
5. hall_ID -> {no_of_seats, hall_name}
6. screening_ID -> {screen_time, screen_date, movie_ID, hall_ID}
7. movie_ID -> {movie_title, movie_director, movie_genre, movie_length}

And it can be clearly stated from all the functional dependencies that the closure of seat_reserve_ID, that is

$(\text{seat\_reserve\_ID})^{+}$ ->R

As there are no multivalued attribute in our database hence the given relation is in 1NF Form.

Now we can clearly see that there are no partial dependencies in our Relation R as partial dependencies occur when a subset of candidate key is capable to derive other non-prime attributes in the relation. But as in our Relation, there is only one attribute

hence there cannot be any proper subset of the candidate key capable to derive other attribute.

Hence this proves that our table is in 2NF Form.

Now, we can clearly notice from the dependencies that there are transitive dependencies present (as booking_ID, screening_ID and seat_ID being non-prime derives another non-prime attributes), hence, the table is not in 3NF Form.

After breaking the functional dependencies such that there are no non-prime to non-prime dependencies (transitive dependencies) table would come in 3NF Form and the functional dependencies now looks like:

1. seat_reserve_ID -> {booking_ID, seat_ID, hall_ID}
2. p_ID -> {p_name, age, phonenumber}
3. booking_ID -> {p_ID, price, screening_ID}
4. seat_ID -> {seat_number, seat_row, hall_ID}

5. hall_ID -> {no_of_seats, hall_name}
6. screening_ID -> {screen_time, screen_date, movie_ID, hall_ID}
7. movie_ID -> {movie_title, movie_director, movie_genre, movie_length}

Now, in the above dependencies we can clearly notice that there are no dependencies that corresponds to a prime attribute (prime attribute is a proper subset of candidate key, in our case
only seat_Reserve_ID is a prime attribute), hence it is safe to mention that our Relation is now in BCNF Form.

Here, the functional dependencies after doing normalization till BCNF Form corresponding to the table present in our database are as followed:

- Functional Dependency (1) corresponds to table Seat Reserved
- Functional Dependency (2) corresponds to table Booking
- Functional Dependency (3) corresponds to table People
- Functional Dependency (4) corresponds to table Screening
- Functional Dependency (5) corresponds to table Hall
- Functional Dependency (6) corresponds to table Movie
- Functional Dependency (7) corresponds to table Seat.

DDL

Creating database and tables:

```
CREATE DATABASE MOVIE_THEATER
USE MOVIE_THEATER
CREATE TABLE Movie
(
movie_ID VARCHAR (5) NOT NULL,
movie_title VARCHAR (20) NOT NULL,
movie_director VARCHAR (20) NOT NULL,
movie_length INT NOT NULL,
movie_genre VARCHAR (15) NOT NULL,
PRIMARY KEY (movie_ID)
);
CREATE TABLE Hall
(
hall_ID VARCHAR (4) NOT NULL,
no_of_seats INT NOT NULL,
hall_name VARCHAR (10) NOT NULL,
PRIMARY KEY (hall_ID)
);
CREATE TABLE People
(
p_ID VARCHAR (10) NOT NULL,
p_name VARCHAR(20) NOT NULL,
age INT NOT NULL,
phonenumber VARCHAR (10) NOT NULL,
PRIMARY KEY (p_ID)
);
CREATE TABLE Screening
(
screening_ID VARCHAR(10) NOT NULL,
```

```
movie_ID VARCHAR(5) NOT NULL,
hall_ID VARCHAR(4) NOT NULL,
screen_time TIME NOT NULL,
screen_date DATE NOT NULL,
PRIMARY KEY (screening_ID),
FOREIGN KEY (movie_ID) REFERENCES Movie(movie_ID),
FOREIGN KEY (hall_ID) REFERENCES Hall(hall_ID)
);
CREATE TABLE Seat
(
seat_ID VARCHAR(7) NOT NULL,
seat_number INT NOT NULL,
seat_row CHAR(1) NOT NULL,
hall_ID VARCHAR(4) NOT NULL,
PRIMARY KEY (seat_ID),
FOREIGN KEY (hall_ID) REFERENCES Hall(hall_ID)
);
CREATE TABLE Booking
(
booking_ID VARCHAR(6) NOT NULL,
price INT NOT NULL,
screening_ID VARCHAR(10) NOT NULL,
p_ID VARCHAR (10) NOT NULL,
PRIMARY KEY (booking_ID),
FOREIGN KEY (screening_ID) REFERENCES Screening(screening_ID),
FOREIGN KEY (p_ID) REFERENCES People(p_ID)
);
CREATE TABLE Seat_Reserved
(
seat_Reserve_ID VARCHAR(6) NOT NULL,
screening_ID VARCHAR(10) NOT NULL,
seat_ID VARCHAR(7) NOT NULL,
```

booking_ID VARCHAR(6) NOT NULL,

PRIMARY KEY (seat_Reserve_ID),

FOREIGN KEY (screening_ID) REFERENCES Screening(screening_ID),

FOREIGN KEY (seat_ID) REFERENCES Seat(seat_ID),

FOREIGN KEY (booking_ID) REFERENCES Booking(booking_ID)

);

EXAMPLES OF INSERTING VALUES

INSERT INTO Movie (movie_ID,movie_title, movie_director,movie_length,movie_genre)

VALUES('M1','Avengers Endgame','Russo Brothers',181,'Superhero');

INSERT INTO Movie (movie_ID,movie_title, movie_director,movie_length,movie_genre)

VALUES('M2','Avatar','James Cameron',162,'Sci-Fi');

INSERT INTO Movie (movie_ID,movie_title, movie_director,movie_length,movie_genre)

VALUES('M3','Titanic','James Cameron',195,'Drama');

```
CREATE TABLE Movie
(
movie_ID VARCHAR (5) NOT NULL,
movie_title VARCHAR (20) NOT NULL,
movie_director VARCHAR (20) NOT NULL,
movie_length INT NOT NULL,
movie_genre VARCHAR (15) NOT NULL,
PRIMARY KEY (movie_ID)
);
INSERT INTO Movie (movie_ID,movie_title, movie_director,movie_length,movie_genre)
VALUES('M1','Avengers Endgame','Russo Brothers',181,'Superhero' );
INSERT INTO Movie (movie_ID,movie_title, movie_director,movie_length,movie_genre)
VALUES('M2','Avatar','James Cameron',162,'Sci-Fi' );
INSERT INTO Movie (movie_ID,movie_title, movie_director,movie_length,movie_genre)
VALUES('M3','Titanic','James Cameron',195,'Drama' );
SELECT * FROM Movie;
```

100 %

Results | Messages

	movie_ID	movie_title	movie_director	movie_length	movie_genre
1	M1	Avengers Endgame	Russo Brothers	181	Superhero
2	M2	Avatar	James Came...	162	Sci-Fi
3	M3	Titanic	James Came...	195	Drama

INSERT INTO Hall(hall_ID,no_of_seats,hall_name)

VALUES('H1',8,'Platinum');

INSERT INTO Hall(hall_ID,no_of_seats,hall_name)

VALUES('H2',6,'EXecutive');

```
INSERT INTO Hall(hall_ID,no_of_seats,hall_name)
VALUES('H3',4,'Royal');

INSERT INTO People(p_ID,p_name,age,phonenumber)
VALUES('P1','Anu',18,'123456789');
INSERT INTO People(p_ID,p_name,age,phonenumber)
VALUES('P2','Bea',20,'123456789');

INSERT INTO Screening (screening_ID,movie_ID,hall_ID,screen_time,screen_date)
VALUES ('SCR1','M1','H3','17:35:00','2020-10-05');
INSERT INTO Screening (screening_ID,movie_ID,hall_ID,screen_time,screen_date)
VALUES ('SCR2','M1','H3','17:35:00','2020-10-06');

INSERT INTO Seat (seat_ID,seat_number,seat_row, hall_ID)
VALUES('H1S1',1,'A', 'H1');
INSERT INTO Seat (seat_ID,seat_number,seat_row, hall_ID)
VALUES('H1S2',2,'A', 'H1');

INSERT INTO Booking VALUES ('B1',1160,'SCR1','P5');
INSERT INTO Seat_Reserved VALUES ('SR1','SCR1','H3S3','B1');
```

EXAMPLES OF ADDING CONSTRAINTS

```
ALTER TABLE People
ADD CONSTRAINT UniqName UNIQUE (p_name);

ALTER TABLE People
ADD CONSTRAINT CheckAge CHECK (age>12);

ALTER TABLE Movie
ADD CONSTRAINT CheckMovieLength CHECK (movie_length>100);

ALTER TABLE Screening
```

```
ADD CONSTRAINT CheckSDate CHECK (screen_date>getdate());

ALTER TABLE Movie
ADD CONSTRAINT MUniq UNIQUE (movie_title);

ALTER TABLE Booking
ADD CONSTRAINT CheckPrice CHECK (price>100);
```

TRIGGERS

CREATING TRIGGER IF ANY UPDATE MADE ON PEOPLE's TABLE:

```
CREATE TABLE People_Audit(
p_ID VARCHAR (10) NOT NULL,
p_name VARCHAR(20) NOT NULL,
age INT NOT NULL,
phonenumber VARCHAR (10) NOT NULL,
Audit_Action varchar(100),
Audit_Timestamp datetime,
PRIMARY KEY (p_ID)
);

CREATE TRIGGER trgAfterInsert
ON People
for UPDATE
AS
begin
	declare @p_ID VARCHAR (10);
	declare @p_name VARCHAR(20);
	declare @age INT;
	declare @phonenumber VARCHAR (10);
	declare @audit_action varchar(100);

	select @p_ID=i.p_ID from inserted i;
	select @p_name=i.p_name from inserted i;
	select @age=i.age from inserted i;
	select @phonenumber=i.phonenumber from inserted i;

	if update(p_name)
```

```
            set @audit_action='Updated Record -- After Update Trigger.';
        if update(phonenumber)
            set @audit_action='Updated Record -- After Update Trigger.';

        insert into
People_Audit(p_ID,p_name,age,phonenumber,Audit_Action,Audit_Timestamp)
        values(@p_ID,@p_name,@age,@phonenumber,@audit_action,getdate());

        PRINT 'AFTER UPDATE Trigger fired.'
end
GO
UPDATE People
SET phonenumber= '456123789'
WHERE p_ID='P3';
```

```
CREATE TRIGGER trgAfterInsert
ON People
for UPDATE
AS
begin
    declare @p_ID VARCHAR (10);
    declare @p_name VARCHAR(20);
    declare @age INT;
    declare @phonenumber VARCHAR (10);
    declare @audit_action varchar(100);

    select @p_ID=i.p_ID from inserted i;
    select @p_name=i.p_name from inserted i;
    select @age=i.age from inserted i;
    select @phonenumber=i.phonenumber from inserted i;

    if update(p_name)
        set @audit_action='Updated Record -- After Update Trigger.';
    if update(phonenumber)
        set @audit_action='Updated Record -- After Update Trigger.';

    insert into People_Audit(p_ID,p_name,age,phonenumber,Audit_Action,Audit_Timestamp)
    values(@p_ID,@p_name,@age,@phonenumber,@audit_action,getdate());

    PRINT 'AFTER UPDATE Trigger fired.'
```

100 %

Messages

```
(1 row affected)
AFTER UPDATE Trigger fired.

(1 row affected)

Completion time: 2020-05-28T09:09:12.6204860+05:30
```

```
CREATE TABLE Booking_Audit(
booking_ID VARCHAR(6) NOT NULL,
price INT NOT NULL,
screening_ID VARCHAR(10) NOT NULL,
p_ID VARCHAR (10) NOT NULL,
Audit_Action varchar(100),
Audit_Timestamp datetime,
PRIMARY KEY (booking_ID)
);

CREATE TRIGGER trgBookingUpdate
ON Booking
for UPDATE
AS
begin
	declare @booking_ID VARCHAR (10);
	declare @price INT;
	declare @screening_ID VARCHAR(10);
	declare @p_ID VARCHAR (10);
	declare @Audit_Action varchar(100);

	select @booking_ID=i.booking_ID from inserted i;
	select @price=i.price from inserted i;
	select @screening_ID=i.screening_ID from inserted i;
	select @p_ID=i.booking_ID from inserted i;

	if update(booking_ID)
		set @audit_action='Updated Record -- After Update Trigger.';
	if update(screening_ID)
		set @audit_action='Updated Record -- After Update Trigger.';
```

```
        insert into
Booking_Audit(booking_ID,price,screening_ID,p_ID,Audit_Action,Audit_Timestamp)
        values(@booking_ID,@price,@screening_ID,@p_ID,@audit_action,getdate());


        PRINT 'AFTER UPDATE Trigger fired.'
end
GO
```

```sql
for UPDATE
AS
begin
    declare @booking_ID VARCHAR (10);
    declare @price INT;
    declare @screening_ID VARCHAR(10);
    declare @p_ID VARCHAR (10);
    declare @Audit_Action varchar(100);

    select @booking_ID=i.booking_ID from inserted i;
    select @price=i.price from inserted i;
    select @screening_ID=i.screening_ID from inserted i;
    select @p_ID=i.booking_ID from inserted i;

    if update(booking_ID)
        set @audit_action='Updated Record -- After Update Trigger.';
    if update(screening_ID)
        set @audit_action='Updated Record -- After Update Trigger.';

    insert into Booking_Audit(booking_ID,price,screening_ID,p_ID,Audit_Action,Audit_Timestamp)
    values(@booking_ID,@price,@screening_ID,@p_ID,@audit_action,getdate());

    PRINT 'AFTER UPDATE Trigger fired.'
end
GO
UPDATE Booking
SET screening_ID= 'SCR5'
WHERE booking_ID='B4';
```

100 %

Messages

```
(1 row affected)
AFTER UPDATE Trigger fired.

(1 row affected)

Completion time: 2020-05-31T11:53:16.4132156+05:30
```

QUERIES

JOIN QUERIES

Q) List the details of the movie that was screened on 2020-10-06.

```
SELECT movie_title, movie_director,movie_length, movie_genre, hall_ID
FROM Movie
LEFT OUTER JOIN Screening
ON Screening.movie_ID =Movie.movie_ID
WHERE screen_date = '2020-10-06';
```

Q) List all the details of the seat of that Executive hall.

```
SELECT seat_ID,seat_row,seat_number
FROM Seat
RIGHT OUTER JOIN Hall
ON Seat.hall_ID=Hall.hall_ID
WHERE Hall.hall_name='EXecutive';
```

CO RELATED -NESTED QUERIES

Q) List the details of the person(s) whose screening_ID is SCR2.

```
SELECT p.p_name,p.phonenumber
FROM People AS p
WHERE EXISTS (SELECT *
FROM Booking AS b
WHERE b.screening_ID='SCR2' AND b.p_ID=p.p_ID);
```

Q) List the details of the movie whose screening length is between 180 and 190.

```
SELECT m.movie_title AS Title, m.movie_director AS Director,
m.movie_genre AS Genre
FROM Movie AS m
```

```
WHERE EXISTS
        (SELECT * FROM Screening AS s
        WHERE (m.movie_length BETWEEN 180 AND 190) AND
m.movie_ID=s.movie_ID);
```

Q) List out the details of the people who will be watching in SCR3.

```
select p.p_name,p.phonenumber
FROM People AS p
where p.p_ID in
(select b.p_ID from Booking as b
where EXISTS (
select * from Seat_Reserved as s
WHERE s.booking_ID=b.booking_ID AND s.screening_ID='SCR3'));
```

Q) Find the details of the person whose name has Hea.

```
select p.p_name,p.phonenumber
FROM People AS p
WHERE p_name LIKE '%Hea%’;
```

Q) Retrieve the count of customers and the movies.

```
select count(p_ID) AS Number_of_Customers from People;
select count(movie_ID) AS Number_of_Movies from Movie;
```

Q) Retrieve the number of times a person has visited the theatres.

```
select p.p_ID as Customer_id,count(*) as number_of_visits
from People as p
group by p.p_ID
having count(*) > 0 ;
```

CONCLUSION

This system provides secure, organized and efficient storage of huge data. One does not have to be physically present at the theatre to get their jobs done.

During my database management course, I have learned about the basics of database design. This project gave me the opportunity to try the skills in practice. While doing this project I gained a deeper understanding on database design and how it can be implemented in real life situations.

It has provided the user to book movie tickets and manager to keep their records all in one place.

With the advent of various online platform like BookMyShow, use type of movie theatre database system, the process became very simple, hustle free and user-friendly.