Entertainment Booking System

Dhirubhai Ambani Institute of Information and Communication Technology
27-11-2021

Database Management System (IT214)

Prepared by Team S7_T7

Aksh Patel : 201901005
Om Patel : 201901208
Vishal Tekwani : 201901217
Nikhil Jain : 201901218

Mentor TA: Manoj Kumar

Table of Contents:

Section 1: SRS

Software Requirements Specification For Entertainment Booking Show

Version 2.0

Prepared by Team S7_T7
Aksh Patel: 201901005
Om Patel: 201901208
Vishal Tekwani: 201901217
Nikhil Jain: 201901218

DAIICT

21/09/2021

❖ Table of Contents For SRS:

(A) Description of the Case Study

1. Purpose

- The purpose of this document is to build an online system to manage ticket booking for entertainment events and to ease the process of ticket booking and it's management.
- It will enable customers to browse through various shows and events. Customers can book their tickets online without any need of physical presence at the booking place.
- The tickets of events such as Sport Matches, Movie Shows and Live Concerts (E.g Music) can be booked.
- The history of the tickets which are previously bought by a particular customer is also maintained and displayed.
- The availability of the tickets for any event is also maintained in an efficient way and retrieved as and when needed by updating it simultaneously.
- System also allows adding new events like book launch, audio launch etc. This system will enable the searching of the ticket for any event in a much more efficient way.

2. Intended Audience and Reading Suggestions

- The intended audience is mainly the youth and the people who like to attend and enjoy fun events and shows.
- Although the tendency to attend these events by the young ones is more but particularly it doesn't target any specific age group.
- It provides many facilities like buying tickets online, saving time, saving money by selecting best offers, safety of money transactions.
- Anyone who knows how to surf online and knows how to book tickets can use this service irrespective of the age limits .
- This document is written for the developers and managers who are interested in our system and customers who want to use our system.
- This system can be used by people who have an active internet connection on their devices.

3. Product Scope

- This software system will function as a ticketing system for entertainment events such as sporting events, film screenings, and live concerts.
- This product allows the customer to purchase a ticket with only one click on their smartphone.
- Also, the scope of this product is bright since with the advent and rise of new technology, everything is moving online, and people don't want to go to a real site merely to make a reservation.
- This system will be built to make it as simple as possible for clients to book tickets, which would otherwise have to be done by standing in long queues at theatres.
- This technology is particularly developed to allow customers to order and cancel tickets online at any time of the day according to their convenience.
- To establish a consistent review procedure, preformatted reply forms are utilised at every stage of the tickets, booked/cancelled passage through the system. A relational database with a list of events, timings, seats, pricing, bookings, and booking history is included in the system.

4. Description

- **Traditional Booking System**

  → People who use conventional methods encounter several problems, such as waiting in lengthy lines, incurring transport costs, managing booking timings, and so on. Even after putting up lots of effort, it is not assured that tickets will still be available.

  → Many times black-marketeer blackmarkets the tickets which leads people to buy tickets at higher prices than expected.

  → If there are multiple shows of the same event then they must go to each location and obtain information from the Ticket Window which takes a long time.

  → Also if you want to cancel your booking then you will not get refunds of the ticket. So there is double loss of money because they have to again spend money to get new tickets.

- **How our System Solves these problems**

  → Our Entertainment Booking System solves all the layman's problems by providing some additional facilities and features through online mode.

  → This system makes the task of preparing and executing an event much simpler by eliminating paperwork ,keeping track of everything and lowering the amount of time and energy you have to spend for booking tickets standing in a line.

  → Musicals, festivals, Concerts, stage plays and even movie theatres all face the same issues as described above. Whether it's a concert tour for a well-known band or just the day-to-day operations of a local movie theatre, the software can make life simpler for the event's management and customers.

- **Additional Features and Facilities**

  → The Customer accesses the Entertainment Booking System for entertainment events through the Internet.

  → This system is available for 24*7 so that customers can visit our system any time and get what they want.

- **System Working**

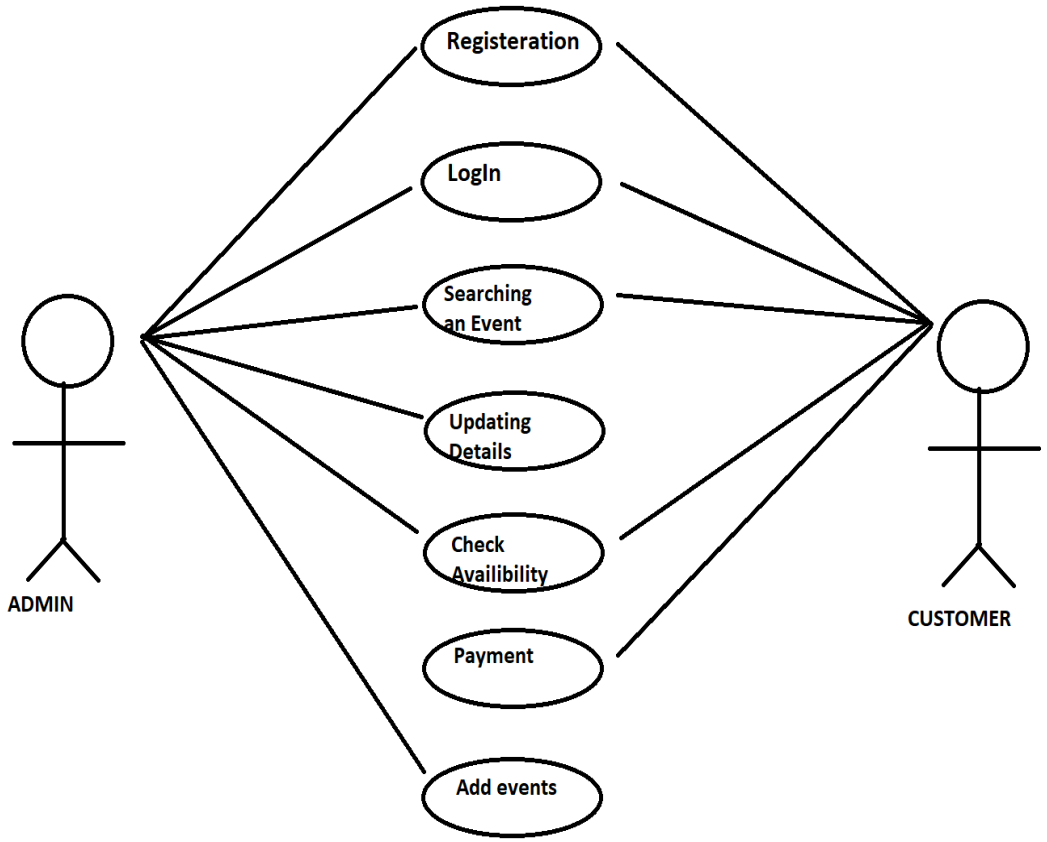
  → The Entertainment Booking System has two active actors, customer and Admin.

  → The system will store different movies, sports matches and concerts' past details and other important details like seat number, seat area, movie details and other details.

  → The customer can login to the system and book/cancel the tickets based on his preferences and seat availability as well as make/receive payment. He can also have access to his booking history.

  → The Admin has direct access to the whole system. Admin can login to the system, add events, modify customer and event information, and update the database after a transaction, and so on.

- **Following figure is a diagrammatic representation of the functionality of Admin and Customer.**



(B) Fact Finding Phase

1. Background Reading

- The product is built on established business concepts such as **bookmyshow.com** and **Paytm.com**.

- **BookmyShow:**
  Features:
  → All kinds of movies are available.
  → The user interface is excellent.
  → System doesn't crash very often.
  → Tickets can be booked at affordable prices.
  → Movie tickets, events, concerts, and sports tickets may all be booked quickly and easily.
  → Simple venue, city, or event search.
  → You have the option of going to a single-screen theatre or a multiplex.
  → The ability to search for movie reviews, trailers, and theatre listings.

Flaws:

→ It doesn't support multiple events.

- **Paytm:**

Features:

→ Has an interactive UI.
→ Has a feature of vouchers and cashbacks also.
→ The cancellation Protect function gives users a better return guarantee when they book tickets through Paytm.

Flaws:

→ The greatest concern is the possibility of identifying theft. Because we are not culturally used to digital transactions, even well-educated people are vulnerable to phishing scams.

- **Requirements**

→ Registration / Login
→ Online searching of different shows
→ Booking tickets at affordable prices
→ Payment Gateway

- **Working/flow of Existing System:**

→ Users first have to login to their website.
→ Then users have to select the event that they want to get information about.
→ Then he can book tickets by making online payments.
→ Then he will get a confirmation mail for tickets.

- **Flaws of the Existing system are as follows:**

→ The major flaw we found is that many systems don't support booking for multiple kinds of events.
→ Since information for all kinds of events is not available it is found very inconvenient to the Users (because they have to search here and there for booking of tickets for multiple events).
→ For example it might not be possible to book movie tickets as well as sport match tickets at the same site.
→ Sometimes when a show is very hit or popular many people start using the website at the same time and due to that servers will get down or the website keeps on buffering/loading.

- **Requirements gathered from background reading:**

→ We need a system which has good browsing speed.
→ We have to allow customers to book tickets for any events not specific to only movies.
→ We should also ensure that their payment is made successful without any risk of fraud.

- **References**:

https://in.bookmyshow.com/
https://www.justdial.com/

2. Interview/s

- **Interview 1:**

Digital Solutions: Role Play Interview Plan

System: bookmyevent

Project Reference: SF/SJ/2021/12

Interviewee: 1. Aksh Patel (Role Play)

Designation: CEO at bookmyevent

Interviewer:

1. Om Patel (Role Play)

   Designation: Business Development Executive-Digital Solutions

2. Vishal Tekwani (Role Play)

   Designation: Developer - Digital Solutions

Date: 05/10/2021 **Time:** 14:30

Duration: 45 minutes **Place:** Online Mode

Purpose of Interview:

Preliminary meeting to identify functionalities and features that can be added to the system.

Agenda:

- Initial Ideas.
- Any specific requirements.
- Issues with the online booking system.
- Managing user experience.

Documents to be brought to the interview:

- Any Survey Documents.
- Any Future Plans.

- ## Interview 2:

Digital Solutions: Role Play Interview Plan

Project Reference: SF/SJ/2021/15

Interviewee: 1. Karan Prajapati

Designation: User

Interviewer:

3. Aksh Patel (Role Play)

   Designation: Business Development Executive - Digital Solutions

4. Nikhil Jain (Role Play)

   Designation: Developer - Digital Solutions

Date: 20/10/2021 **Time:** 7:30

Duration: 45 minutes **Place:** Online Mode

Purpose of Interview:

To identify problems with the traditional system of buying tickets and issues with the existing system.

Agenda:

- Problems in the existing system.
- Improvement in security(during transaction).
- Expected Functionalities.
- Any improvement needed.

- ## Interview 3:

Digital Solutions: Role Play Interview Plan

System: BookYourEvent

Project Reference: SF/SJ/2003/12

Interviewee: 1) Nikhil Jain (Role Play)

Designation: CEO of BookYourEvent

Interviewer: 1) Om Patel

Designation: Business Development Executive - Digital Solutions

2) Vishal Tekwani

Designation: Developer - Digital Solutions

Date: 05/10/2021 **Time:** 17:30

Duration: 45 minutes **Place:** Online Mode

Purpose of Interview:

Preliminary meeting to identify security issues and requirements for their system and infrastructure.

Agenda:

- Problems with security and any other concerns
- Current security procedures
- Initial ideas
- Follow-up actions

Documents to be brought to the interview:

- Rough plan of infrastructure required.
- Any documents relating to current security procedures and issues.

Requirements gathered from interviews:

Issues with online booking system:

- Data redundancy and inconsistency.
- Difficulty in accessing data.
- Uncontrolled concurrent access leading to inconsistency.
- Data isolation and integrity issues.

Problems with the traditional system of buying tickets:

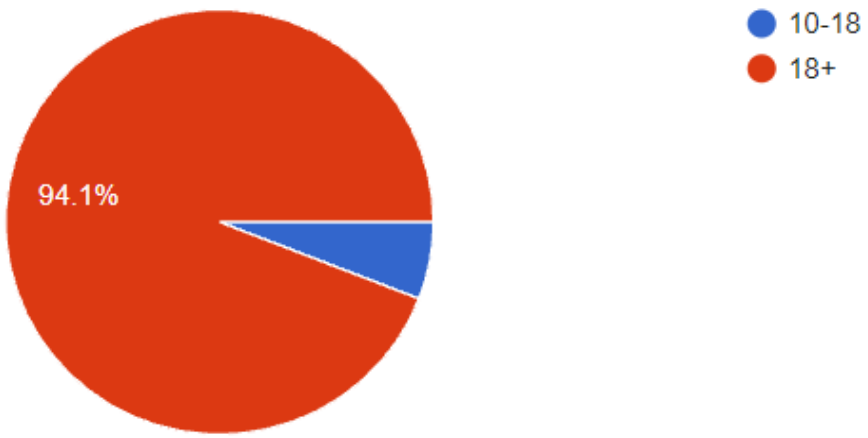
- Time consuming as the customer has to wait in lengthy lines and manage booking timings.
- Incurring transport costs.
- Black marketing.
- No refund on cancellation.

Problems with current security procedures:

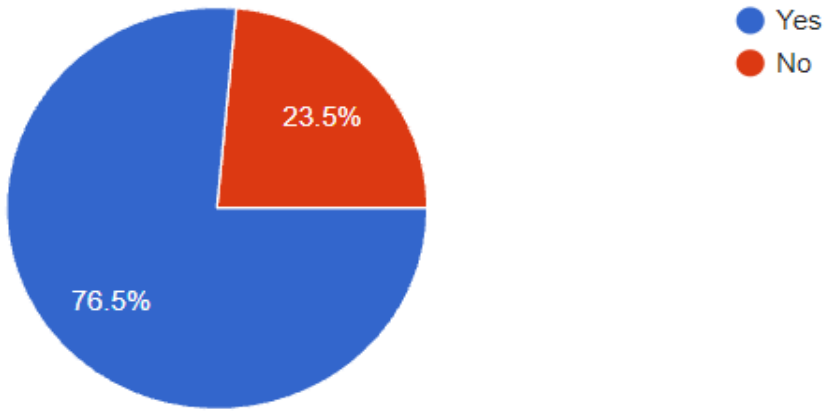
Hard to provide data access to some but not all.

3. Questionnaire/s and Summary of Responses

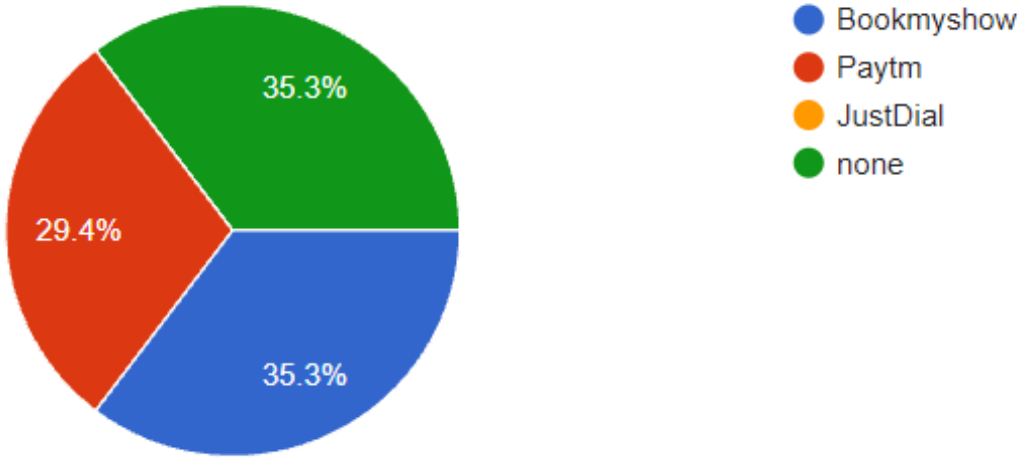
1. **Which age group do you belong to**
    a. **10-18**
    b. **18+**



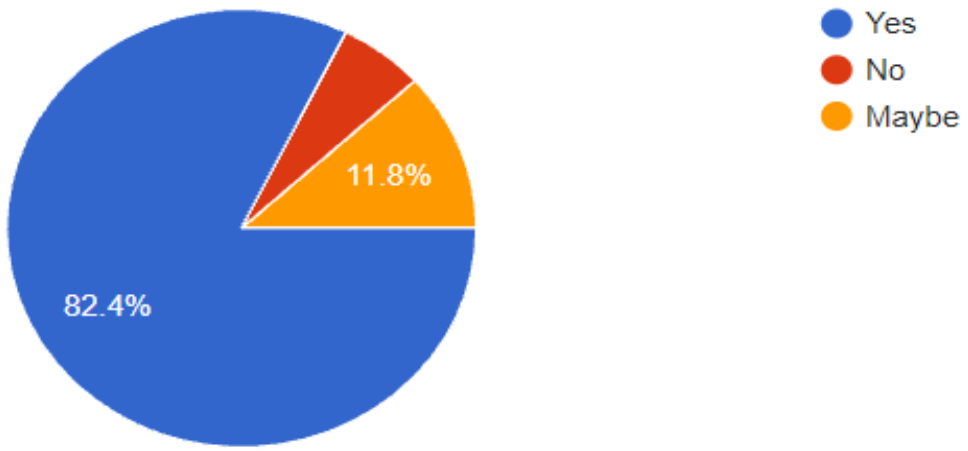
2. **Do you book the tickets online ?**
    a. **Yes**
    b. **No**



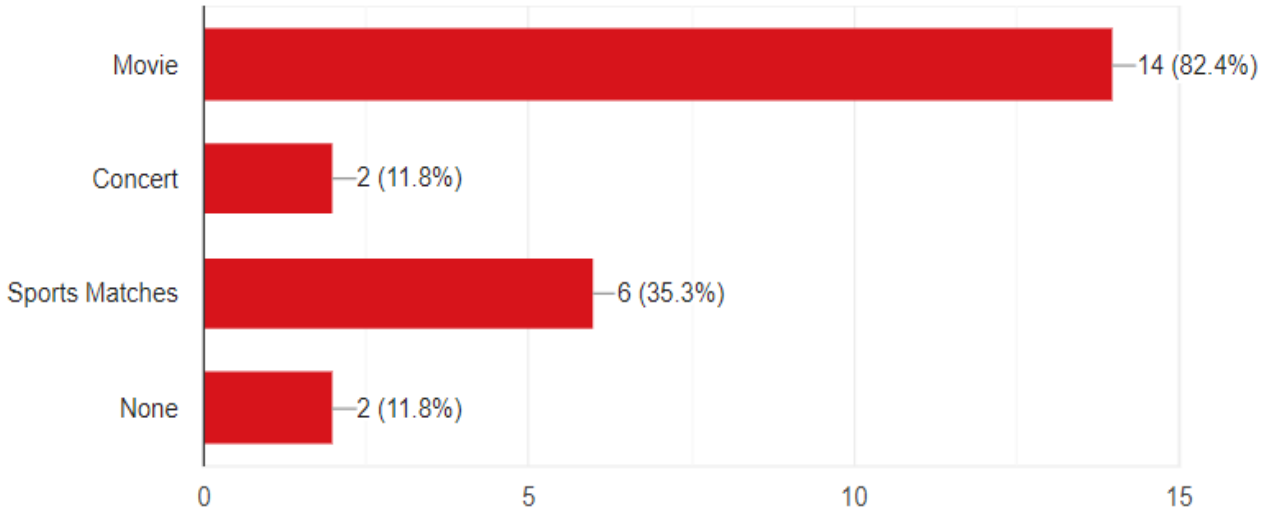
3. **Which application do you use for booking?**
    a. **Bookmyshow**
    b. **Paytm**
    c. **JustDial**



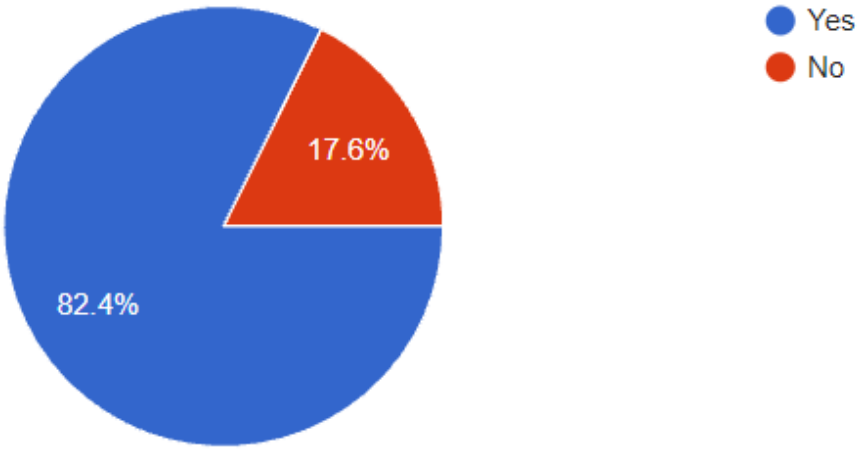
4. **Do you want an application which contains all booking platforms in one ?**
    a. **Yes**
    b. **No**



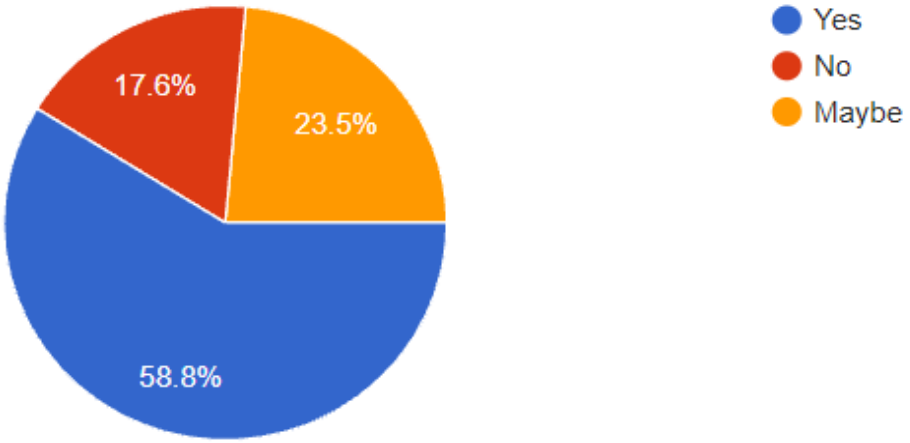
5. **For which events do you generally buy tickets online?**
    a. **Movie**
    b. **Concert**
    c. **Sports Matches**
    d. **None**



6. **Do you find it more convenient in comparison to the traditional method of booking?**
   a. **Yes**
   b. **No**

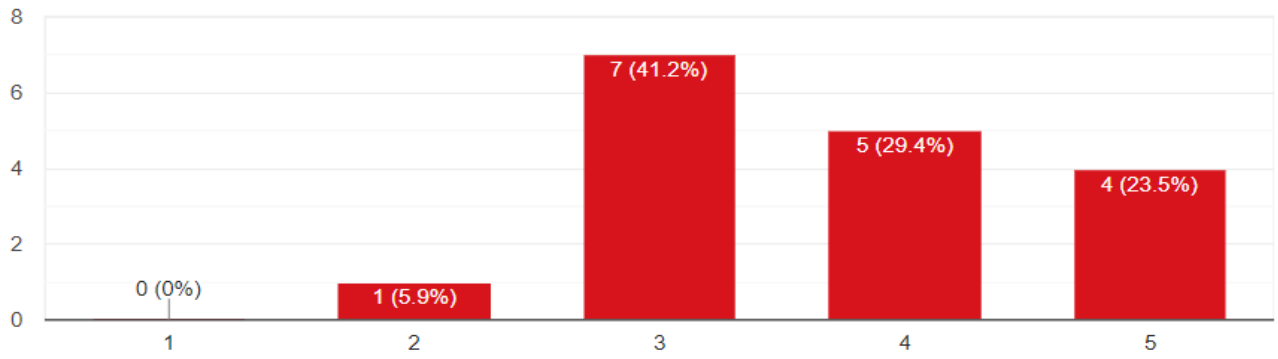


7. **Do you want any updates regarding movies or concerts?**
   a. **Yes**
   b. **No**
   c. **Maybe**



8. **Are you satisfied with the current application you are using for booking tickets?**

Not satisfied 1 2 3 4 5 Fully Satisfied



9. **Do you want to give any special suggestions**

Following are some suggestions from users:

- Software must be easy to use.
- This software must have enough capacity to handle multiple customers at a time.
- Add features for customer review in the system.
- It has a low data consumption rate.
- This software must have a good security system to prevent any malicious attack or any leaking of private information such as credit or debit card details.
- It must have a record for past customers and provide discounts for those customers.

4. Observation/s

IT Solutions: Observations

System: PVR Cinemas

Project Reference: SF/SJ/2021/20

Observations by: Nikhil Jain (Digital Solutions- Company employee)

Date: 14/8/2003 **Time:** 14:30

Duration: 45 minutes **Place:** BookYourEvent

Observations:

The following things are reviewed and observed at the PVR Cinemas and noted down as follows :

1. It is observed that cinemas require more manpower because they need a person who sells tickets and also settles disputes among people standing in queues.
2. Our system must require a huge capacity for customers because at a time many customers can login into our system. So, our system doesn't crash if such a thing happens.
3. The process for booking tickets consumed a lot of time as the customers had to wait in long queues and also no refund was provided on cancellation of tickets.
4. The customers were also not given the opportunity to select the exact seat number and area according to their convinience.
5. Also Cinemas have fixed timings for ticket windows which further reduces the user experience as they can't book tickets when they are free.
6. Because the entire system needed to be maintained by hand, the process of storing, preserving, and retrieving data was very laborious and time-consuming.
7. If we do not maintain a database then storing physical records and stuff is very tedious and difficult to maintain.
8. They do not maintain the list of old customers and also the records were never kept in a logical sequence.
9. Managing the event is also difficult because we don't know ahead of time how many tickets will get sold.

(C). Fact Finding Chart:

Objective	Technique	Subject	Time commitment
To understand the features, working flow and flaws of an existing system	Background reading	Websites	2 days
To find issues with the online booking system	Interview	CEO of bookmyevent	45 minutes
To find out concerns regarding the traditional booking system	Interview	User	45 minutes
To find out problems with the current security procedures	Interview	CEO of BookYourEvent	45 minutes
To understand the specific requirements of the user and gather their suggestions.	Questionnaire	Civilians	Half day

(D) Requirements

1. Security of data.
2. Data Accuracy.
3. Assurance of Safety of User's data.
4. Greater efficiency of the software.
5. User Friendliness and interactiveness.
6. Proper distribution of privileges.
7. The system should have ACID properties (Atomicity, Consistency, Isolation, Durability).
8. The Admin should be able to login, add events, search events, update details and check availability.
9. The User should also be able to make payment.

(E) User Classes and Characteristics

1. **Admin**
   → Admin has the right to register/login to the database, add events, search events, update details of the events and the customers and check availability from the database.
   → It has the highest privileges among all sets of User classes available.
   → It also assigns the roles to the event managers about the particular event which he/she is organising.

2. **Event Managers**
   → Admin gives the event organiser special powers in the role of event manager, who is in charge of a certain event.
   → Event manager has more privileges than the User but less than that of the Admin of the software.
   → Event manager has rights to change the happenings of a certain event only (which is assigned to him).
   → E.g changing the time of the event, Updating the value, Adding, Removing etc. of the events to which his role is being assigned to.

3. **Civilians (Users)**
   → Civilians can get information like dates, venue, prices of tickets, etc of various events like sport matches, movies, concerts. They will get facilities for cancelling tickets.
   → The privileges they obtain is the least (i.e less than that of the admin as well as the event manager).
   → They can book the event, cancel the event, search for the events, make payment for the particular event and can check the history of bookings and the transaction status.

(F) Operating Environment

1. Hardware, Software or Connectivity Requirements:

A. Hardware Requirements:

- A device with a working internet connection.
- Should have a device which supports the latest software updates.

B. Software Requirements:

- More compatible operating systems like windows 8 or 10 or 11 for windows.
- Web Browser: Google Chrome, Mozilla Firefox or Windows Edge.
- Should have an account on apps like paytm for online payment transactions.

2. External Interface Requirements:

- **Hardware Interfaces**

  If we expand in the future, we may require our own server to store information in the form of cache, or numerous server sets in different locations of different cities to distribute load.

- **Software Interfaces**

  The website server will use a sql query to retrieve data from our database and display it. A domain name will be required to execute the web application online. To access the online application, the user will need to use any browser.

(G) Product Functions:

- Program data independence.
- Efficient data access.
- Data integrity and security.
- Concurrent access and crash recovery.
- Data administration.
- Reduced Application Development Time.

(H) Privileges

There are majorly three types of groups :

1. **Admin**
   → Admin have the rights to Registration/Login, Add Events, Search events, Update details, Check availability, Check Status.
   → Admin has access to every command of Data Definition Language (DDL) and Data Manipulation Language (DML).

2. **Event Manager**
   → Event manager has rights to update or modify information of his/her events.

3. **Users**
   → Users can get information about various events at any time he wants.
   → Users may buy tickets for events like concerts, sport matches, and many more by picking the seat number and area according to their convenience and can also check the history of their bookings.

(I) Assumptions

- A Google account is used by the user.
- This programme is being used with a good internet connection by the user.
- All of the system(Hardware/Software) requirements outlined in the preceding section are met by the user's device.
- A Browser , such as Chrome or Firefox, has been installed by users.
- The Database company will have server response 24x7.
- Server has enough capability to handle the large amount of requests from users.
- User is familiar with the working environment of the website.

(J) Business Constraints

- Using this database Businessmen get to know in which area we have to increase our service online so more customers can use this service.
- The project is scheduled to be completed in three months, although it may take longer if a serious problem is discovered.
- We should also have a separate copy of our complete database because cases might arise of failure or crashing of the server.
- Saving the customer details and history of bookings is limited to 30 days due to less data capacity storage. After 30 days the data is automatically deleted.
- Databases should be regularly updated so that users can see the latest information.

Section2: Noun Analysis

1. Noun (& Verb) Analysis:

Table 1: All Extracted Noun and Verbs

Nouns	Verbs
black-marketeer	preparing
Ticket	executing
Concerts	providing
Festivals	managing
Customer	Receive
Admin	make
window	access
Movie	update
system	modify
tickets	providing
time	eliminating
problem	lowering
entertainment	executing
Database	booking
track	keeping
paperwork	refunds
Theatre	incurring
stage	waiting
Money	working
People	managing
Tour	cancel
event	visit
software	expected
actors	described
Event details	based
Event manager	
Events booked	

2. Truncating Initial Noun List:

Table 2: Accepted Noun and Verbs List

Candidate entity set	Candidate attribute set	Candidate relationship set
People(customer)	customer_id, customer_name, customer_email, mob_no	Books, searches
Event manager	manager_id, manager_name, event_type_id, manager_email, mob_no	Manages
Event(Event_type)	event_type_id, event_name, no_of_shows	Manages, searches
Event_details	event _id, event_name, event_type_id, event_date, time, total_tickets, left_tickets	Manages, searches
Events_booked	id, event_id, event_manager_id, customer_id, tickets_booked	books

Table 3: Rejected noun and verb list

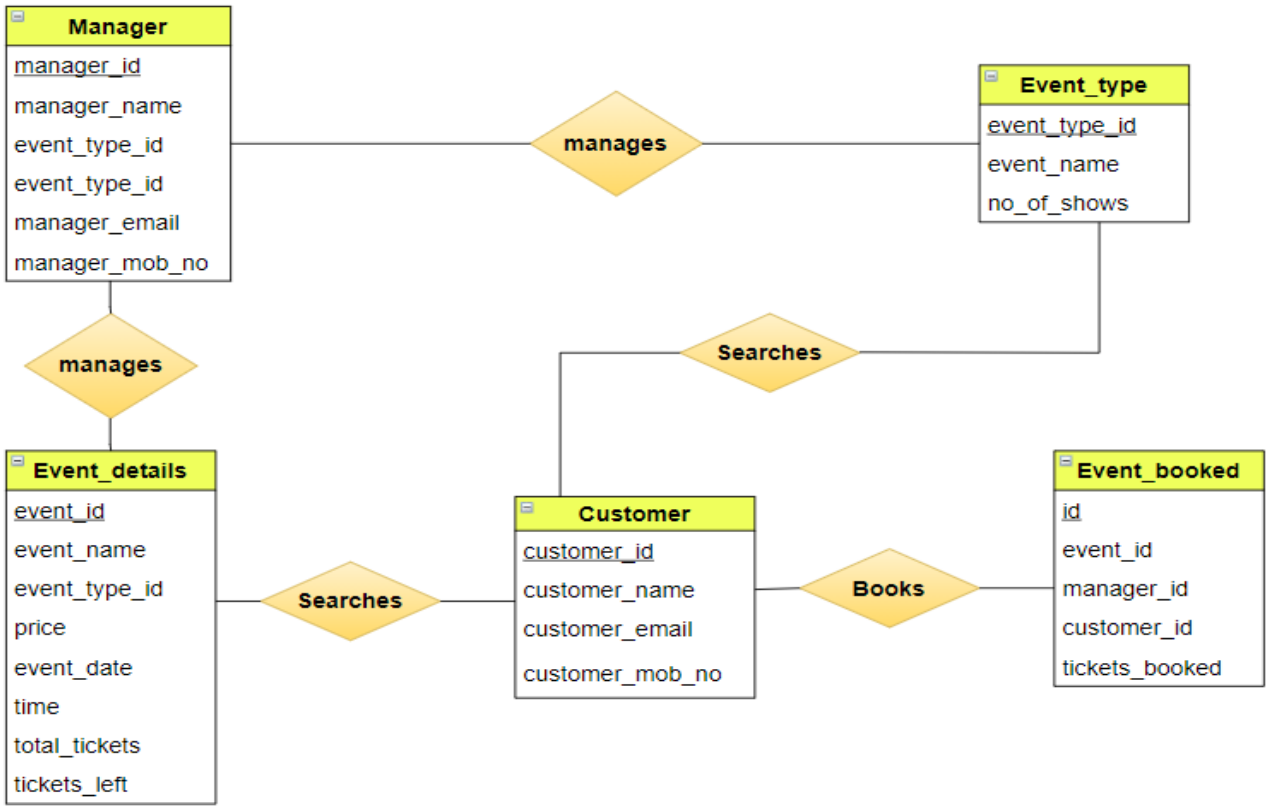
Nouns	Reject Reason
software	Duplicate
black-marketeer	Irrelevant
festivals	Irrelevant
actors	Vague
entertainment	General
window	Irrelevant
Paperwork	General

Tour	Irrelevant
Theatre	General
track	Irrelevant
Time	Irrelevant
Money	general
Problem	general

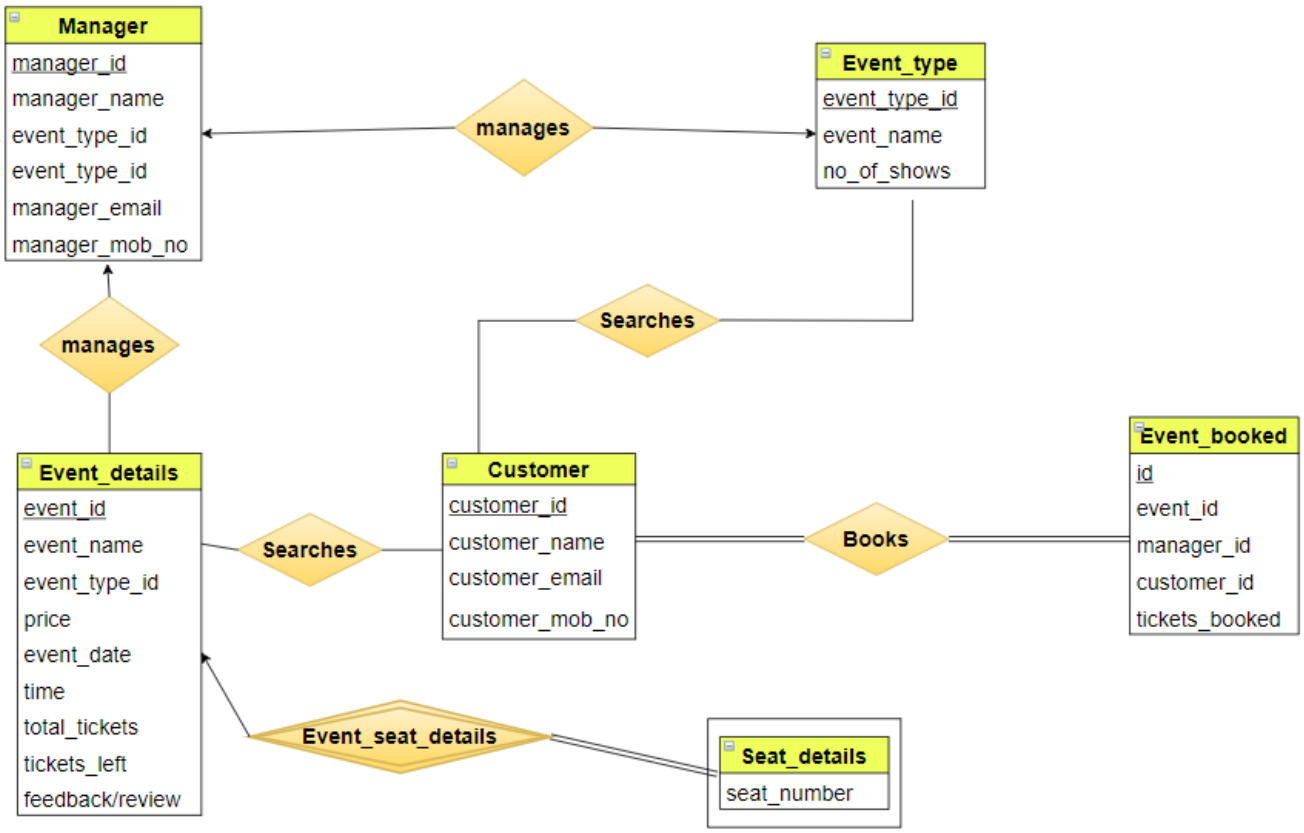
Verbs	**Reject Reason**
Visit	Irrelevant
executing	duplicate
providing	General
preparing	Irrelevant
Lowering	Vague
described	General

Section3: ER Diagram all versions

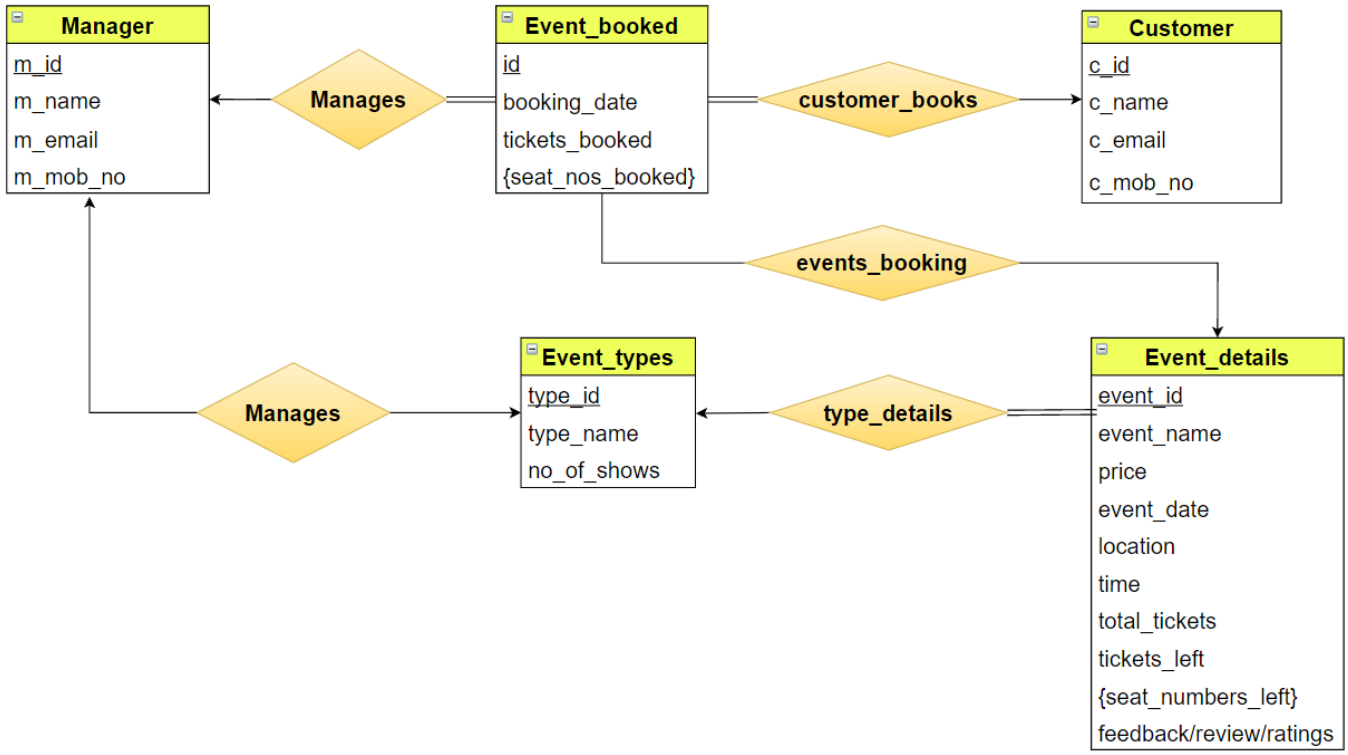
- **ER Diagram Version 1**



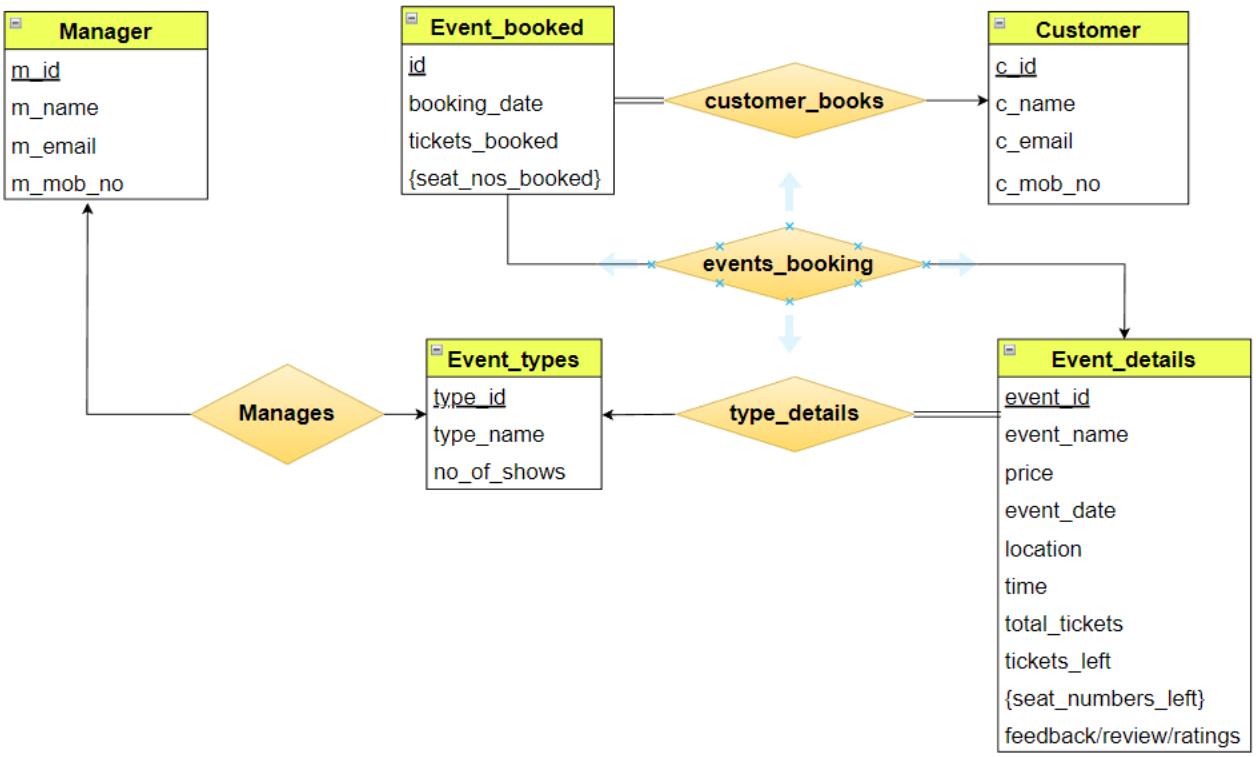
- **ER Diagram Version 2:**



- **ER Diagram Version 3:**



- **Final Version of ER Diagram:**



Section4
Conversion of Final ER-Diagram to Relational Model.

1. Mapping ER Model to Relational Model

→ **Manager(<u>m_id</u>,** m_name, m_email, m_mob_no**)**

→ **Event_booked(<u>id,</u>** c_id, event_id, booking_date, tickets_booked**)**

→ **Event_details(event_id,** type_id, event_name, price, event_date, location, event_time, duration, total_tickets, tickets_left, rating)

→ **Event_types(<u>type_id</u>,** m_id, type_name, no_of_shows**)**

→ **Customer(<u>c_id</u>**, c_name, c_email, c_mob_no**)**

→ **Seats_left(<u>event_id</u>**, seat_no**)**

→ **Seats_booked(<u>id</u>**, seat_no**)**

2. Create DDL Scripts

```
CREATE TABLE manager
(
        m_id int NOT NULL,
        m_name varchar(30),
        m_email varchar(30),
        m_mob_no bigint,
        PRIMARY KEY(m_id)
)
```

```
CREATE TABLE event_booked
(
       id int NOT NULL,
       m_id int,
       c_id int,
       event_id int,
       booking_date date,
       tickets_booked int,
       PRIMARY KEY(id),
       FOREIGN KEY(c_id) REFERENCES customer,
       FOREIGN KEY(event_id) REFERENCES event_details
)
```

```
CREATE TABLE event_details
(
        event_id int NOT NULL,
        type_id int,
        event_name varchar(30),
        price int,
        event_date data,
        location varchar(30),
        event_time time,
        duration int,
        total_tickets int,
        tickets_left int,
        rating int,
```

```
    check(rating<=5 and rating>=1)
    PRIMARY KEY(event_id),
    FOREIGN KEY(type_id) REFERENCES event_type
)
```

```
CREATE TABLE event_type
(
    type_id int NOT NULL,
    m_id int,
    type_name int,
    no_of_shows int,
    PRIMARY KEY(type_id),
    FORIEGN KEY(m_id) REFERENCES manager
)
```

```
CREATE TABLE customer
(
    c_id int NOT NULL,
    c_name varchar(30),
    c_email varchar(30),
    c_mob_no bigint,
    PRIMARY KEY(c_id)
)
```

```
CREATE TABLE seats_left
(
    event_id int NOT NULL,
    seat_no int,
    PRIMARY KEY(event_id,seat_no)
)
```

```
CREATE TABLE seat_booked
(
    id int NOT NULL,
    seat_no int,
    PRIMARY KEY(id,seat_no)
)
```

Section5: Normalization and Schema Refinement

I. Normalization & Schema Refinement

1. List all the Relations & Schemas with all details (Original Design of Database)

→ **Manager(<u>m_id</u>**, m_name, m_email, m_mob_no)
→ **Event_booked(<u>id</u>**, c_id, event_id, booking_date, tickets_booked)
→ **Event_details(<u>event_id</u>**, type_id, event_name, price, event_date, location, event_time, duration, total_tickets, tickets_left, rating)
→ **Event_types(<u>type_id</u>**, m_id, type_name, no_of_shows)
→ **Customer(<u>c_id</u>**, c_name, c_email, c_mob_no)
→ **Seats_left(<u>event_id</u>**, **<u>seat_no</u>**)
→ **Seats_booked(<u>id</u>**, **<u>seat_no</u>**)

2. Identify and list all types of dependencies (PK, FK, Functional Dependencies) for each relation.

→ Manager:

Primary key: m_id
Foreign Key: NULL
m_id → m_name,
m_id → m_email,
m_id → m_mob_no

→ Event_booked:

Primary key: id
Foreign Key: c_id, event_id
id → c_id
id → event_id
id → booking_date
id → tickets_booked

→ Event_details:

Primary key: event_id
Foreign Key: type_id
event_id → type_id,
event_id → event_name,
event_id → price,
event_id → event_date,
event_id → location,
event_id → event_time,
event_id → duration,
event_id → total_tickets,
event_id → tickets_left,
event_id → rating,

→ Event_types:

Primary key: type_id
Foreign Key: NULL

type_id → m_id,
type_id → type_name,
type_id → no_of_shows

→ Customer:

Primary key: c_id
Foreign Key: NULL
c_id → c_name,
c_id → c_email,
c_id → c_mob_no

→ Seats_left:

Primary Key: event_id, seat_no
Foreign Key: event_id

→ Seats_booked

Primary Key: id, seat_no
Foreign Key: id

3. Investigate every schema for the following:

- **List of redundancies existing for every schema which is part of the database.**
  → No redundancy is there in our schemas.

- **List of update, delete, and insert anomalies for every schema.**
  → No insert, delete and update anomalies.

4. Normalize the database up to 1NF (scalar values).

→ We have multi-valued attribute (seat_no) in 2 entity-sets (event_booked and event_types) which is resolved by breaking them into 2 tables.
→ Also we don't have any composite attributes.

5. Normalize the database further to 2NF (Remove Partial Dependencies).

→ No partial dependencies are present because all the primary keys are mostly single attributes. (A proper subset of CK → NPA is not present).

6. Identify List of redundancies existing for the schema in 2NF.

→ No redundancies.

7. Normalize it further to 3NF/BCNF (Remove Transitive Dependencies)

→ No Transitive dependencies because no non-key attribute is dependent on any other non-key attribute.

8. Normalize it further to BCNF.

→ It is already in BCNF because every determinant is a super key in all the dependencies.

II. Re-write DDL Scripts.

1. Recreate database by writing all Create Table statements (DDL) to accommodate the new design which is in 3NF/BCNF (removing your original version of relations).
2. Define appropriate constraints of all types (domain, PK, FK, Referential) for these tables.
3. Create an instance of this new database by populating it using appropriate INSERT INTO statements /using scripts. Make sure that every table has at least 80-100 tuples.

List all the Relations & Schemas with all details (Final Design of Database)

→ **Manager(<u>m_id</u>**, m_name, m_email, m_mob_no)
→ **Event_booked(<u>id</u>**, c_id, event_id, booking_date, tickets_booked)
→ **Event_details(<u>event_id</u>**, type_id, event_name, price, event_date, location, event_time, duration, total_tickets, tickets_left, rating)
→ **Event_types(<u>type_id</u>**, m_id, type_name, no_of_shows)
→ **Customer(<u>c_id</u>**, c_name, c_email, c_mob_no)
→ **Seats_left(<u>event_id</u>**, **<u>seat_no</u>**)
→ **Seats_booked(<u>id</u>**, **<u>seat_no</u>**)

DDL Scripts

```
CREATE TABLE manager
(
    m_id int NOT NULL,
    m_name varchar(30),
    m_email varchar(30),
    m_mob_no bigint,
    PRIMARY KEY(m_id)
)
```

- **No. of records = 4**

```
105
106 select * from manager
107
```

Messages | Query History | Explain | Notifications | Data Output

	m_id [PK] integer	m_name character varying (30)	m_email character varying (30)	m_mob_no bigint
1	1	Aksh Patel	201901005@daiict.ac.in	9289839293
2	2	Om Patel	201901208@daiict.ac.in	9282939283
3	3	Vishal Tekwani	201901217@daiict.ac.in	9373828492
4	4	Nikhil Jain	201901218@daiict.ac.in	9273827382

```
CREATE TABLE event_booked
(
    id int NOT NULL,
    m_id int,
    c_id int,
    event_id int,
    booking_date date,
    tickets_booked int,
    PRIMARY KEY(id),
    FOREIGN KEY(c_id) REFERENCES customer,
    FOREIGN KEY(event_id) REFERENCES event_details
)
```

- **No. of records = 80**

```
SELECT * FROM event_booked
```

Data Output | Explain | Messages | Notifications

	id [PK] integer	c_id integer	event_id integer	booking_date date	tickets_booked integer
1	1	1	1	2021-01-01	1
2	2	2	2	2021-01-02	1
3	3	3	3	2021-01-03	1
4	4	4	4	2021-01-04	1
5	5	5	5	2021-01-05	1
6	6	6	6	2021-01-06	1
7	7	7	7	2021-01-07	1
8	8	8	8	2021-01-08	1
9	9	9	9	2021-01-09	1
10	10	10	10	2021-01-10	1

```
CREATE TABLE event_details
(
    event_id int NOT NULL,
    type_id int,
    event_name varchar(30),
    price int,
    event_date data,
    location varchar(30),
    event_time time,
    duration int,
    total_tickets int,
    tickets_left int,
    rating int,
    check(rating<=5 and rating>=1)
    PRIMARY KEY(event_id),
    FOREIGN KEY(type_id) REFERENCES event_type
)
```

- **No. of records = 100**

```
106 select * from event_details
107
```

Messages | Query History | Explain | Notifications | Data Output

	event_id [PK] integer	type_id integer	event_name character varying (100)	price integer	event_date date	location character varying (30)	event_time time without time zone	duration integer	tot int
98	98	4	Officers In The Portal	455	2021-03-24	Bangalore	02:00:00	2	
99	99	4	Hunters Of The Worlds	266	2021-11-12	Jaipur	02:00:00	3	
100	100	4	Assassins And Children	210	2021-07-14	Ahmedabad	10:00:00	2	

```
CREATE TABLE event_type
(
    type_id int NOT NULL,
    m_id int,
    type_name int,
    no_of_shows int,
    PRIMARY KEY(type_id),
    FOREIGN KEY(m_id) REFERENCES manager
)
```

- **No. of records = 4**

```
106 select * from event_type
107
```

Messages | Query History | Explain | Notifications | Data Output

	type_id [PK] integer	m_id integer	type_name character varying (30)	no_of_shows integer
1	1	1	movies	40
2	2	2	sports	20
3	3	3	concerts	10
4	4	4	audio_launch	30

```
CREATE TABLE customer
(
    c_id int NOT NULL,
    c_name varchar(30),
    c_email varchar(30),
    c_mob_no bigint,
    PRIMARY KEY(c_id)
)
```

- **No. of records = 80**

```
106 select * from customer
107
```

Messages | Query History | Explain | Notifications | Data Output

	c_id [PK] integer	c_name character varying (30)	c_email character varying (30)	c_mob_no bigint
77	77	Aria Henderson	Wendy@gmail.com	8600288821
78	78	Kylan Perkins	Aspen@gmail.com	8734648594
79	79	Brianna Stephenson	Cali@gmail.com	8869008367
80	80	Jazmyn Espinoza	Rayne@gmail.com	9003368140

```
CREATE TABLE seats_left
(
   event_id int NOT NULL,
   seat_no int,
   PRIMARY KEY(event_id,seat_no),
   FOREIGN KEY(event_id) REFERENCES event_details
)
```

- **No. of records = 668**

```
106 select * from seats_left
107
```

Messages | Query History | Explain | Notifications | Data Output

	event_id [PK] integer	seat_no [PK] integer
665	100	2
666	100	3
667	100	4
668	100	5

```
CREATE TABLE seat_booked
(
   id int NOT NULL,
   seat_no int,
   PRIMARY KEY(id,seat_no),
   FOREIGN KEY(id) REFERENCES event_booked
)
```

- **No. of records = 80**

```
select * from seat_booked
```

Messages | Query History | Explain | Notifications | Data Output

	id [PK] integer	seat_no [PK] integer
77	77	1
78	78	1
79	79	1
80	80	1

Insert queries from csv file:

```
COPY main_db.manager (m_id,m_name,m_email,m_mob_no)
FROM 'D:\DBMS Lab\Lab8\manager.csv'
DELIMITER ',' CSV HEADER;

COPY main_db.event_type (type_id, m_id, type_name, no_of_shows)
FROM 'D:\DBMS Lab\Lab8\event_type.csv'
DELIMITER ',' CSV HEADER;

COPY main_db.customer (c_id,c_name,c_email,c_mob_no)
FROM 'D:\DBMS Lab\Lab8\customer.csv'
DELIMITER ',' CSV HEADER;

COPY main_db.event_details
(event_id,type_id,event_name,price,event_date,location,event_time,duration,total_tickets,tickets_left,rating)
FROM 'D:\DBMS Lab\Lab8\event_details.csv'
DELIMITER ',' CSV HEADER;

COPY main_db.event_booked (id,c_id,event_id,booking_date,tickets_booked)
FROM 'D:\DBMS Lab\Lab8\event_booked.csv'
DELIMITER ',' CSV HEADER;

COPY main_db.seats_left (event_id,seat_no)
FROM 'D:\DBMS Lab\Lab8\seats_left.csv'
DELIMITER ',' CSV HEADER;

COPY main_db.seat_booked (id,seat_no)
FROM 'D:\DBMS Lab\Lab8\seat_booked.csv'
DELIMITER ',' CSV HEADER;
```

SECTION-6

SQL: Final DDL Scripts, Insert statements, 40 SQL Queries with Snapshots of output of each query.

→ SQL Queries:

1. Find number of shows of event_type ‘sports’.

select no_of_shows from event_type
where type_name='sports'

```
-- 1. Find number of shows of event_type ‘sports’.
select no_of_shows from event_type
where type_name='sports'
```

Messages | Query History | Explain | Notifications | Data Output

	no_of_shows integer
1	20

No. of tuples = 1

2. Find all the events in Ahmedabad.

select * from event_details
where location='Ahmedabad'

```
-- 2. Find all the events in Ahmedabad.
select * from event_details
where location='Ahmedabad'
```

Messages | Query History | Explain | Notifications | Data Output

	event_id [PK] integer	type_id integer	event_name character varying (100)	price integer	event_date date	location character varying (30)	event_time time without time zone	duration integer	total_tickets integer	tickets_left integer	rating integer
1	1	1	Commander Of Moondust	177	2021-01-22	Ahmedabad	06:00:00	3	8	7	
2	8	1	Guest Of Our Ship	412	2021-11-25	Ahmedabad	11:00:00	2	5	4	
3	11	1	Children Of New Worlds	337	2021-03-03	Ahmedabad	10:00:00	1	9	8	
4	21	1	Foreigners And Children	368	2021-06-10	Ahmedabad	10:00:00	3	9	8	
5	33	1	Favor Of The Eyes	261	2021-03-14	Ahmedabad	01:00:00	2	9	9	
6	100	4	Assassins And Children	210	2021-07-14	Ahmedabad	10:00:00	2	5	5	

No. of tuples = 6

3. Finding the event_name having the maximum price.

select event_name from event_details
where price = (select max(price) from event_details)

```
-- 3. Finding the event_name having the maximum price.
select event_name from event_details
where price = (select max(price) from event_details)
```

Messages | Query History | Explain | Notifications | Data Output

	event_name character varying (100)
1	World Series

No. of tuples = 1

4. Find all events on a particular date.

select event_name from event_details
where event_date = '2021-01-22'

```
-- 4. Find all events on a particular date.
select event_name from event_details
where event_date = '2021-01-22'
```

Messages | Query History | Explain | Notifications | Data Output

	event_name character varying (100)
1	Commander Of Moondust

No. of tuples = 1

5. Find all the events in the month of January.

select event_name from event_details
where extract (month from event_date) = 1

```
-- 5. Find all the events in the month of January.
select event_name from event_details
where extract (month from event_date) = 1
```

Messages | Query History | Explain | Notifications | Data Output

	event_name character varying (100)
1	Commander Of Moondust
2	Foreigners Of Moondust
3	Pilots Of Our Culture
4	Hidden By The Stars
5	The Masters
6	Wimbledon
7	Olympic Games.
8	Amit Trivedi
9	A Promised Land by Barac...
10	Moby Dick by Herman Mel...

No. of tuples = 10

6. Find the number of events lasting more than 2hrs.

select count(event_name) from event_details
where duration>2

```
-- 6. Find the number of events having duration more than 2hrs.
select count(event_name) from event_details
where duration>2
```

Messages | Query History | Explain | Notifications | Data Output

	count bigint
1	31

No. of tuples = 1

7. Find the event details having ratings more than 3.

select * from event_details
where rating>3

```
-- 7. Find the event details having ratings more than 3.
select * from event_details
where rating>3
```

Messages Query History Explain Notifications Data Output

	event_id [PK] integer	type_id integer	event_name character varying (100)	price integer	event_date date	location character varying (30)	event_time time without time zone	duration integer	total_tickets integer	tickets_left integer	rating integ
32	79	4	Harry Potter	348	2021-10-21	Jaipur	04:00:00	3	10	9	
33	83	4	Don Quixote by Miguel de ...	235	2021-06-03	Jaipur	05:00:00	2	9	8	
34	84	4	One Hundred Years of Solit...	375	2021-07-24	Bangalore	11:00:00	3	10	9	
35	88	4	Ronels journey to cto	385	2021-06-08	Jaipur	01:00:00	3	10	9	
36	89	4	Alchemist	331	2021-09-03	Gurgaon	07:00:00	1	6	5	
37	93	4	Defender Of Our Ship	348	2021-07-19	Gurgaon	07:00:00	1	6	6	
38	94	4	Leader In The Beginning Of...	337	2021-09-02	Jaipur	10:00:00	2	10	10	
39	98	4	Officers In The Portal	455	2021-03-24	Bangalore	02:00:00	2	5	5	
40	99	4	Hunters Of The Worlds	266	2021-11-12	Jaipur	02:00:00	3	6	6	

No. of tuples = 40

8. Find event_name where number of left tickets is more than 5

select event_name,total_tickets,tickets_left from event_details
where tickets_left>5

```
-- 8. Find event_name where number of left tickets is more than 5
select event_name,total_tickets,tickets_left from event_details
where tickets_left>5
```

Messages Query History Explain Notifications Data Output

	event_name character varying (100)	total_tickets integer	tickets_left integer
59	Don Quixote by Miguel de ...	9	8
60	One Hundred Years of Solit...	10	9
61	The Great Gatsby by F	7	6
62	Ronels journey to cto	10	9
63	Half girlfriend	8	7
64	Flames	6	6
65	Medic Of Stardust	6	6

No. of tuples = 65

9. Find details of manager whose name is Aksh Patel

select * from manager
where m_name='Aksh Patel'

```
-- 9. Find details of manager whose name is Aksh Patel
select * from manager
where m_name='Aksh Patel'
```

Messages | Query History | Explain | Notifications | Data Output

	m_id [PK] integer	m_name character varying (30)	m_email character varying (30)	m_mob_no bigint
1	1	Aksh Patel	201901005@daiict.ac...	9289839293

No. of tuples = 1

10. Find total price of all the tickets of all events

select sum(total_tickets*price) from event_details

```
-- 10. Find total price of all the tickets of all events
select sum(total_tickets*price) from event_details
```

Messages | Query History | Explain | Notifications | Data Output

	sum bigint
1	226441

No. of tuples = 1

11. Find name of manager of event_type ‘movies’.

select m_name from manager
natural join event_type
where type_name='movies'

```
-- 11. Find name of manager of event_type ‘movies’
select m_name from manager
natural join event_type
where type_name='movies'
```

Messages | Query History | Explain | Notifications | Data Output

	m_name character varying (30)
1	Aksh Patel

No. of tuples = 1

12. Find the manager name of the event ‘Doctors Of The Ocean’.

select m_id,m_name from manager
natural join event_type
natural join event_details
where event_name = 'Doctors Of The Ocean'

```
-- 12. Find the manager id and name of the event ‘Doctors Of The Ocean’.
select m_id,m_name from manager
natural join event_type
natural join event_details
where event_name = 'Doctors Of The Ocean'
```

Messages | Query History | Explain | Notifications | Data Output

	m_id [PK] integ	m_name character varying (30)
1	1	Aksh Patel

No. of tuples = 1

13. Find the net total worth of all the event types and print in decreasing order

select type_name,sum(total_tickets*price) as net_worth from event_details
natural join event_type
group by type_id,type_name
order by net_worth desc

```
-- 13.  Find the net total worth of all the event types
select type_name,sum(total_tickets*price) as net_worth from event_details
natural join event_type
group by type_id,type_name
order by net_worth desc
```

Messages | Query History | Explain | Notifications | Data Output

	type_name character varying (30)	net_worth bigint
1	movies	84532
2	audio_launch	71060
3	sports	42320
4	concerts	28529

No. of Tuples = 4

14. Find the cheapest event of type ‘concert’ in Bangalore.

select event_name from event_details
where price = (select min(price) from event_details
natural join event_type
where location = 'Bangalore' and
type_name = 'concerts')

```
-- 14. Find the cheapest event of type ‘concert’ in Bangalore.
select event_name from event_details
where price = (select min(price) from event_details natural join event_type
                    where location = 'Bangalore' and type_name = 'concerts')
```

Messages | Query History | Explain | Notifications | Data Output

	event_name character varying (100)
1	Amit Trivedi

No. of Tuples = 1

15. Create a trigger function which is called when any insert is done in the event_booked table which updates tickets_left in event_details.

```
CREATE OR REPLACE FUNCTION Trigger_Function()
RETURNS trigger
LANGUAGE 'plpgsql'
AS
$BODY$
BEGIN
        UPDATE event_details set tickets_left = tickets_left - new.tickets_booked
        where event_id = new.event_id;
        raise notice 'Avaliable tickets are updated....';
        return new;
end
$BODY$;
```

```
CREATE TRIGGER Trigger_Total_Tickets_Update
AFTER INSERT
ON event_booked
FOR EACH ROW
WHEN (pg_trigger_depth()<1)--!
EXECUTE FUNCTION Trigger_Function();
```

```
INSERT into event_booked(id,c_id,event_id,booking_date,tickets_booked)
values(81,1,1,'18-11-2021',1);
```

```
-- 15. Create a trigger which calls when any insert is done in the event_booked table
--      which updates tickets left in event_details.
CREATE OR REPLACE FUNCTION Trigger_Function()
RETURNS trigger
LANGUAGE 'plpgsql'
AS
$BODY$
BEGIN
    UPDATE event_details set tickets_left = tickets_left - new.tickets_booked
    where event_id = new.event_id;
    raise notice 'Avaliable tickets are updated....';
    return new;
end
$BODY$;
```

```
CREATE TRIGGER Trigger_Total_Tickets_Update
AFTER INSERT
ON event_booked
FOR EACH ROW
WHEN (pg_trigger_depth()<1)--!
EXECUTE FUNCTION Trigger_Function();

insert into event_booked(id,c_id,event_id,booking_date,tickets_booked)
values(81,1,1,'18-11-2021',1);
```

Messages | Query History | Explain | Notifications | Data Output

```
NOTICE:  Avaliable tickets are updated....
INSERT 0 1

Query returned successfully in 82 msec.
```

Output:

```
SELECT * FROM event_booked
```

Messages | Query History | Explain | Notifications | Data Output

	id [PK] integer	c_id integer	event_id integer	booking_date date	tickets_booked integer
81	81	1	1	2021-11-18	1

Tickets_left of event_id = 1 is updated from 7 to (7-1) = 6.

Messages | Query History | Explain | Notifications | Data Output

event_id [PK] integer	type_id integer	event_name character varying (100)	price integer	event_date date	location character varying (30)	event_time time without time zone	duration integer	total_tickets integer	tickets_left integer
99	4	Hunters Of The Worlds	266	2021-11-12	Jaipur	02:00:00	3	6	6
100	4	Assassins And Children	210	2021-07-14	Ahmedabad	10:00:00	2	5	5
1	1	Commander Of Moondust	177	2021-01-22	Ahmedabad	06:00:00	3	8	6

No. of Tuples Updated = 1

16. Find customers who had booked most tickets.

```
create or replace view Tickets_Booked as
select sum(tickets_booked) as total from event_booked
group by c_id
```

```
select c_name from customer
natural join event_booked
group by c_id,c_name
having sum(tickets_booked) = (select max(total) from Tickets_Booked)
```

```
-- 16. Find customers who had booked most tickets.
create or replace view Tickets_Booked as
select sum(tickets_booked) as total from event_booked
group by c_id

select c_name,sum(tickets_booked) as Tickets_Booked from customer
natural join event_booked
group by c_id,c_name
having sum(tickets_booked) = (select max(total) from Tickets_Booked)
```

Messages | Query History | Explain | Notifications | Data Output

	c_name character varying (30)	tickets_booked bigint
1	acob Gould	2

No. of Tuples = 1

17. Update email of manager ‘Aksh Patel’ to ‘ap12345@gmail.com’

UPDATE manager
set m_email = 'ap12345@gmail.com'
where m_name = 'Aksh Patel'

```
-- 17. Update email of manager ‘Aksh Patel’ to ‘ap12345@gmail.com’
UPDATE manager
set m_email = 'ap12345@gmail.com'
where m_name = 'Aksh Patel'
```

Messages | Query History | Explain | Notifications | Data Output

	m_id [PK] integer	m_name character varying (30)	m_email character varying (30)	m_mob_no bigint
1	2	Om Patel	201901208@daiict.ac.in	9282939283
2	3	Vishal Tekwani	201901217@daiict.ac.in	9373828492
3	4	Nikhil Jain	201901218@daiict.ac.in	9273827382
4	1	Aksh Patel	ap12345@gmail.com	9289839293

No. of Tuples Affected = 1

18. Update name of event ‘Sharma bhajan’ to ‘SB Group’

UPDATE event_details
set event_name = 'SB Group'
where event_name = 'Sharma bhajan'

```
-- 18. Update name of event ‘Sharma bhajan’ to ‘SB Group’
UPDATE event_details
set event_name = 'SB Group'
where event_name = 'Sharma bhajan'
```

Messages | Query History | Explain | Notifications | Data Output

	event_id [PK] integer	type_id integer	event_name character varying (100)	price integer	event_date date	location character varying (30)
77	79	4	Harry Potter	348	2021-10-21	Jaipur
78	66	3	SB Group	495	2021-09-22	Jaipur

No. of Tuples Affected = 1

19. Provide event wise sum of prices of tickets in that event

SELECT type_name,SUM(total_tickets*price) FROM event_type NATURAL JOIN event_details group by type_id

```
--19. Provide event wise sum of prices of tickets in that event
SELECT type_name,SUM(total_tickets*price) FROM event_type
NATURAL JOIN event_details group by type_id
```

Data Output | Explain | Messages | Notifications

	type_name character varying (30)	sum bigint
1	audio_launch	71060
2	sports	42320
3	concerts	28529
4	movies	84532

No.of Tuples = 4

20. Find the total sum of amount paid by each customer

SELECT c_name,SUM(tickets_booked*price) as amount_paid FROM event_booked
NATURAL JOIN customer NATURAL JOIN
event_details group by c_name

```
--20. Find the total sum of amount paid by each customer
SELECT c_name,SUM(tickets_booked*price) as amount_paid FROM event_booked
NATURAL JOIN customer NATURAL JOIN
event_details group by c_name
```

Data Output | Explain | Messages | Notifications

	c_name character varying (30)	amount_paid bigint
1	Kira Mccann	330
2	Howard Austin	173
3	Lizbeth Dunn	478
4	Jamiya Flores	424
5	Finley Hardin	107
6	Libby Lang	185
7	Gage Rush	348
8	Angeline Villanueva	286

No.of tuples = 80

21. Provide the number of tickets left on particular date in increasing order

SELECT event_date,SUM(tickets_left) FROM event_details
group by event_date order by event_date

```
--21. Provide the number of tickets left on particular date in increasing order
SELECT event_date,SUM(tickets_left) FROM event_details
group by event_date order by event_date
```

Data Output | Explain | Messages | Notifications

	event_date date	sum bigint
1	2021-01-01	9
2	2021-01-04	15
3	2021-01-12	5
4	2021-01-14	4
5	2021-01-18	6
6	2021-01-21	12
7	2021-01-22	6
8	2021-01-23	9

No.of tuples = 76

22. Find a name of the manager in decreasing order according to the number of events handled by them .

SELECT m_name,COUNT(*) FROM event_type NATURAL JOIN manager NATURAL JOIN event_details group by m_name order by COUNT(*) DESC

```
--22. Find a name of the manager in decreasing order according to the number of
--    events handled by them .
SELECT m_name,COUNT(*) FROM event_type NATURAL JOIN manager
NATURAL JOIN event_details group by m_name
order by COUNT(*) DESC
```

Data Output | Explain | Messages | Notifications

	m_name character varying (30)	count bigint
1	Aksh Patel	40
2	Nikhil Jain	30
3	Om Patel	20
4	Vishal Tekwani	10

No.of tuples = 4

23. Find the name of the event where the number of tickets booked is maximum.

SELECT event_name FROM event_details
where total_tickets-tickets_left =
(SELECT max(total_tickets-tickets_left) FROM event_details)

```
--23. Find the name of the event all where the number of tickets booked is maximum.
SELECT event_name FROM event_details
where total_tickets-tickets_left =
(SELECT max(total_tickets-tickets_left) FROM event_details)
```

Data Output | Explain | Messages | Notifications

	event_name character varying (100)
1	Commander Of Moondust

No.of tuples = 1

24. Find the total number of tickets booked by each customer in decreasing order.

```
CREATE OR REPLACE FUNCTION find_no_of_tickets_booked()
returns table (c_name varchar(30),no_of_tickets_booked bigint)
language 'plpgsql'
as
$body$
BEGIN
RETURN QUERY EXECUTE format ('SELECT c_name,COUNT(*) as no_of_tickets_booked FROM customer
NATURAL JOIN event_booked NATURAL JOIN event_details
group by c_name ORDER BY COUNT(*) DESC');
END
$body$
```

```
SELECT "main_db".find_no_of_tickets_booked();
```

```
--24. Find total number of tickets booked by each customer in decreasing order

CREATE OR REPLACE FUNCTION find_no_of_tickets_booked()
returns table (c_name varchar(30),no_of_tickets_booked bigint)
language 'plpgsql'
as
$body$
BEGIN
RETURN QUERY EXECUTE format ('SELECT c_name,COUNT(*) as no_of_tickets_booked FROM customer
NATURAL JOIN event_booked NATURAL JOIN event_details
group by c_name ORDER BY COUNT(*) DESC');
END
$body$
SELECT "main_db".find_no_of_tickets_booked();
```

Data Output | Explain | Messages | Notifications

	find_no_of_tickets_booked record
1	("acob Gould",2)
2	("Howard Austin",1)
3	("Lizbeth Dunn",1)
4	("Jamiya Flores",1)
5	("Finley Hardin",1)
6	("Libby Lang",1)
7	("Gage Rush",1)
8	("Angeline Villanueva",1)

No.of tuples = 80

25. Insert a new customer to our database

INSERT INTO customer
values(81,'Alex Dod','aldo@gmail.com',1293021982)

```
--25. Insert a new customer to our database
INSERT INTO customer values(81,'Alex Dod','aldo@gmail.com',1293021982)
```

Data Output Explain Messages Notifications

```
INSERT 0 1

Query returned successfully in 186 msec.
```

No.of tuples affected=1

26. Insert a new manager to our database

INSERT INTO manager
values(5,'Eric Kumar','erikaa@gmail.com',3209129823)

```
--26. Insert a new manager to our database
INSERT INTO manager values(5,'Eric Kumar','erikaa@gmail.com',3209129823)
```

Data Output Explain Messages Notifications

```
INSERT 0 1

Query returned successfully in 73 msec.
```

No.of tuples affected = 1

27. Print details of customers who have booked sport events.

select customer.c_name, customer.c_email, customer.c_mob_no
from event_booked natural join customer natural join event_details natural join event_type
where event_type.type_name = 'sports'

```
--27. print details of customers who have booked sport events
select customer.c_name, customer.c_email, customer.c_mob_no
from event_booked natural join customer natural join event_details natural join event_type
where event_type.type_name = 'sports'
```

Data Output

	c_name character varying (30)	c_email character varying (30)	c_mob_no bigint
1	Charlie Blackwell	Bethany@gmail.com	2789193324
2	Isai Burns	Courtney@gmail.com	3955520289
3	Lia Miller	Haven@gmail.com	2052069970
4	Kamron Stafford	Izabella@gmail.com	4506711487
5	Madalyn Clark	Ashlynn@gmail.com	3613771113
6	Howard Austin	Janiya@gmail.com	5730103583
7	Marcos Collier	Tatiana@gmail.com	1732230836

No. of tuples = 20

28. Print names of movies having tickets_left greater than 7

select event_details.event_name , event_details.tickets_left
from event_details natural join event_type
where event_details.tickets_left > 7 and event_type.type_name='movies'

```
--28. print names of movies having tickets_left greater than 7
select event_details.event_name , event_details.tickets_left
from event_details natural join event_type
where event_details.tickets_left > 7 and event_type.type_name='movies'
```

Data Output

	event_name character varying (100)	tickets_left integer
1	Officer Of Space	8
2	Children Of New Worlds	8
3	Officers Of The Worlds	8
4	Men Of Life	9
5	Pilots Of Our Culture	9
6	Emperors And	9
7	Emperors And Commanders	9

No. of tuples = 17

29. Print names of customers who have booked concerts which have rating more than 3

select customer.c_name,event_details.event_name,event_details.rating
from event_booked natural join customer natural join event_details natural join event_type
where event_type.type_name='concerts' and event_details.rating>3

```
--29. print names of customers who have booked concerts which have rating more than 3
select customer.c_name,event_details.event_name,event_details.rating
from event_booked natural join customer natural join event_details natural join event_type
where event_type.type_name='concerts' and event_details.rating>3
```

Data Output

	c_name character varying (30)	event_name character varying (100)	rating integer
1	Jamiya Flores	Darshan raval	5
2	Misael Nolan	Amit Trivedi	4
3	Aleah Banks	Badshah	5
4	Issac Hines	Honey singh	4

No. of tuples = 4

30. Print number of tickets booked of different type of shows on date 2021-12-22

select event_type.type_name,count(event_id)
from event_details natural join event_type
where event_date='2021-12-22'
group by event_type.type_name

```
--30. print number of tickets booked of different type of shows on date 2021-12-22
select event_type.type_name,count(event_id)
from event_details natural join event_type
where event_date='2021-12-22'
group by event_type.type_name
```

Data Output

	type_name character varying (30)	count bigint
1	movies	1

No. of tuples = 1

31. Print manager details managing different event types

select manager.m_name , manager.m_email , manager.m_mob_no , event_type.type_name
from manager natural join event_type

```
--31. print manager details managing different event types
select manager.m_name , manager.m_email , manager.m_mob_no , event_type.type_name
from manager natural join event_type
```

Data Output

	m_name character varying (30)	m_email character varying (30)	m_mob_no bigint	type_name character varying (30)
1	Aksh Patel	201901005@daiict.ac.in	9289839293	movies
2	Om Patel	201901208@daiict.ac.in	9282939283	sports
3	Vishal Tekwani	201901217@daiict.ac.in	9373828492	concerts
4	Nikhil Jain	201901218@daiict.ac.in	9273827382	audio_launch

No. of tuples = 4

32. Print number of events managed by a particular manager (Vishal Tekwani)

select manager.m_name ,event_type.type_name , event_type.no_of_shows
from manager natural join event_type
where manager.m_name='Vishal Tekwani'

```
--32. Print number of events managed by a particular manager (Vishal)
select manager.m_name ,event_type.type_name , event_type.no_of_shows
from manager natural join event_type
where manager.m_name='Vishal Tekwani'
```

Data Output

	m_name character varying (30)	type_name character varying (30)	no_of_shows integer
1	Vishal Tekwani	concerts	10

No. of tuples = 1

33. Provide the average price of all the event types.

select event_type.type_name,avg(price)
from event_details natural join event_type
group by event_type.type_name

```
--33. Provide the event type name average price.
select event_type.type_name,avg(price)
from event_details natural join event_type
group by event_type.type_name
```

Data Output | Explain | Messages | Notifications

	type_name character varying (30)	avg numeric
1	concerts	349.5000000000000000
2	audio_launch	330.2000000000000000
3	movies	275.4750000000000000
4	sports	306.2500000000000000

Successfully run. Total query runtime: 126 msec. 4 rows affected.

No. of tuples = 4

34. Find the total number of tickets left of a particular event type in descending order

select event_type.type_name,sum(tickets_left) as no_of_tickets_left
from event_details natural join event_type
group by event_type.type_name
order by sum(tickets_left) desc

```
--34. Find the total number of tickets left of a particular event type in descending order
select event_type.type_name,sum(tickets_left) as no_of_tickets_left
from event_details natural join event_type
group by event_type.type_name
order by sum(tickets_left) desc
```

Data Output | Explain | Messages | Notifications

	type_name character varying (30)	no_of_tickets_left bigint
1	movies	278
2	audio_launch	197
3	sports	122
4	concerts	70

Successfully run. Total query runtime: 97 msec. 4 rows affected.

No. of tuples = 4

35. Find all managers managing more than 10 events.

select manager.m_id , manager.m_name , event_type.no_of_shows
from manager natural join event_type
where event_type.no_of_shows>10

```
--35. Find all managers managing more than 10 events.
select manager.m_id , manager.m_name , event_type.no_of_shows
from manager natural join event_type
where event_type.no_of_shows>10
```

Data Output | Explain | Messages | Notifications

	m_id integer	m_name character varying (30)	no_of_shows integer
1	1	Aksh Patel	40
2	2	Om Patel	20
3	4	Nikhil Jain	30

Successfully run. Total query runtime: 105 msec. 3 rows affected.

No. of tuples = 3

36. Provide a customer-manager pair for each customer.

select manager.m_name, customer.c_name
from manager natural join event_type natural join event_details natural join event_booked natural join customer

```
--36. Provide a customer-manager pair for each customer.
select manager.m_name, customer.c_name
from manager natural join event_type natural join event_details natural join event_booked natural join customer
```

Data Output | Explain | Messages | Notifications

	m_name character varying (30)	c_name character varying (30)
1	Aksh Patel	acob Gould
2	Aksh Patel	Cristal Dickerson
3	Aksh Patel	Finley Hardin
4	Aksh Patel	Quinn May
5	Aksh Patel	Skylar Pennington

Successfully run. Total query runtime: 88 msec. 81 rows affected.

No. of tuples = 81

37. Find email ids of customers of a particular event (Girl Of The Void).

select customer.c_email
from customer natural join event_booked natural join event_details
where event_details.event_name='Girl Of The Void'

```
--37 Find email ids of customers of particular event (Girl Of The Void).
select customer.c_email
from customer natural join event_booked natural join event_details
where event_details.event_name='Girl Of The Void'
```

Data Output | Explain | Messages | Notifications

	c_email character varying (30)
1	Maria@gmail.com

Successfully run. Total query runtime: 86 msec. 1 rows affected.

No. of tuples = 1

38. Provide customer names and their email managed by manager 'Om Patel'

select customer.c_name , customer.c_email
from manager natural join event_type natural join event_details natural join event_booked natural join customer
where manager.m_name='Om Patel'

```
--38 Provide customer names and their email managed by manager Om
select customer.c_name , customer.c_email
from manager natural join event_type natural join event_details natural join event_booked natural join customer
where manager.m_name='Om Patel'
```

Data Output | Explain | Messages | Notifications

	c_name character varying (30)	c_email character varying (30)
1	Charlie Blackwell	Bethany@gmail.com
2	Isai Burns	Courtney@gmail.com
3	Lia Miller	Haven@gmail.com
4	Kamron Stafford	Izabella@gmail.com
5	Madalyn Clark	Ashlynn@gmail.com

Successfully run. Total query runtime: 91 msec. 20 rows affected.

No. of tuples = 20

39. Find the total number of events in every location.(using group by)

select event_details.location , count(*) as number_of_events
from event_details
group by event_details.location

```
select event_details.location , count(*) as number_of_events
from event_details
group by event_details.location
```

Data Output | Explain | Messages | Notifications

	location character varying (30)	number_of_events bigint
1	Gurgaon	13
2	Jaipur	17
3	Dharamshala	3
4	Ambala	1
5	Ludiana	1

Successfully run. Total query runtime: 123 msec. 21 rows affected.

No. of tuples = 21

40. Increase price of all the tickets by 20% whose rating is >4

update event_details
set price=price+0.2*price
where rating>3

```
--40. Increase price of all the tickets by 20% whose rating is >4
update event_details
set price=price+0.2*price
where rating>3
```

Data Output | Explain | Messages | Notifications

```
UPDATE 40

Query returned successfully in 75 msec.
```

No. of Tuples Affected = 40

Section7: Project Code with output screenshots

Full code:

```
import time
import pymonetdb
import psycopg2

connection = psycopg2.connect(host="localhost",database="Entertainment Booking
System",user="postgres",password="admin")

cursor = connection.cursor()
cursor.execute('SELECT version()')

#display the PostgreSQL database server version
# db_version = cursor.fetchone()
# print(db_version)

# Query 1
ch="Insert into main_db.manager values(6,'Api Kumar','apiaa@gmail.com',3209529823)"
print(ch)
cursor.execute(ch)
connection.commit()


# Query 2
ch="select * from main_db.manager order by m_id"
print(ch)
cursor.execute(ch)
rows=cursor.fetchall()
for r in rows:
   print(r[0],r[1],r[2],r[3])


# Query 3
city=input('Enter the name of the city where you want to find all the events: ')
ch="select * from main_db.event_details where location='"+city+"'"
cursor.execute(ch)
rows=cursor.fetchall()
for r in rows:
   print("event_id = ",r[0], " type_id= ",r[1] , "event_name= ",r[2] ,"price= ",r[3] ,"event_date= ",r[4] ,"location= ",r[5]
,"event_time= ",r[6],"duration= ",r[7] ,"total_tickets= ",r[8] ,"tickets_left= ",r[9] ,"rating= ",r[10])

# Query 4
while(1):
   print('----------------------------------------')
   print('Welcome to the Menu...')
```

```
print('1: Managers')
print('2: Customers')
print('3: event_type')
print('4: event_details')
print('5: events_booked')
print('6: exit')
print('-------------------------------------')
choice = int(input('Enter your choice:'))
print('-------------------------------------')
if choice == 1:
    print('Manager details')
    ch="select * from main_db.manager order by m_id"
    cursor.execute(ch)
    rows=cursor.fetchall()
    for r in rows:
        for c in r:
            print(c,end='  ')
        print()
elif choice == 2:
    print('customers details')
    ch="select * from main_db.customer"
    cursor.execute(ch)
    rows=cursor.fetchall()
    for r in rows:
        for c in r:
            print(c,end='  ')
        print()
elif choice == 3:
    print('event_types details')
    ch="select * from main_db.event_type"
    cursor.execute(ch)
    rows=cursor.fetchall()
    for r in rows:
        for c in r:
            print(c,end='  ')
        print()
elif choice == 4:
    print('event_details details')
    ch="select * from main_db.event_details"
    cursor.execute(ch)
    rows=cursor.fetchall()
    for r in rows:
        for c in r:
            print(c,end='  ')
        print()
```

```
elif choice == 5:
    print('event_booked details')
    ch="select * from main_db.event_booked"
    cursor.execute(ch)
    rows=cursor.fetchall()
    for r in rows:
        for c in r:
            print(c,end='  ')
        print()
else:
    print('Thank you for visiting...')
    print()
    break
```

Queries:

1. Inserting into the manager table.

```
# Query 1
ch="Insert into main_db.manager values(6,'Api Kumar','apiaa@gmail.com',3209529823)"
print(ch)
cursor.execute(ch)
connection.commit()
```

```
104  SELECT * FROM manager
105
```

Messages Query History Explain Notifications Data Output

	m_id [PK] integer	m_name character varying (30)	m_email character varying (30)	m_mob_no bigint
1	2	Om Patel	201901208@daiict.ac.in	9282939283
2	3	Vishal Tekwani	201901217@daiict.ac.in	9373828492
3	4	Nikhil Jain	201901218@daiict.ac.in	9273827382
4	1	Aksh Patel	ap12345@gmail.com	9289839293
5	5	Eric Kumar	erikaa@gmail.com	3209129823
6	6	Api Kumar	apiaa@gmail.com	3209529823

2. Printing the manager table

```
# Query 2
ch="select * from main_db.manager order by m_id"
print(ch)
cursor.execute(ch)
rows=cursor.fetchall()
for r in rows:
   print(r[0],r[1],r[2],r[3])
```

```
PS C:\Users\Lenovo> python -u "d:\DBMS Lab\Lab10\main.py"
select * from main_db.manager order by m_id
1 Aksh Patel ap12345@gmail.com 9289839293
2 Om Patel 201901208@daiict.ac.in 9282939283
3 Vishal Tekwani 201901217@daiict.ac.in 9373828492
4 Nikhil Jain 201901218@daiict.ac.in 9273827382
5 Eric Kumar erikaa@gmail.com 3209129823
6 Api Kumar apiaa@gmail.com 3209529823
PS C:\Users\Lenovo>
```

3. Print all the events in the user given city.

```
# Query 3
city=input('Enter the name of the city where you want to find all the events: ')
ch="select * from main_db.event_details where location='"+city+"'"
cursor.execute(ch)
rows=cursor.fetchall()
for r in rows:
   print("event_id = ",r[0], " type_id= ",r[1] , "event_name= ",r[2] ,"price= ",r[3] ,"event_date= ",r[4] ,"location= ",r[5] ,"event_time= ",r[6],"duration= ",r[7] ,"total_tickets= ",r[8] ,"tickets_left= ",r[9] ,"rating= ",r[10])
```

Output:

```
PS C:\Users\Lenovo> python -u "d:\DBMS Lab\Lab10\main.py"
Enter the name of the city where you want to find all the events: Ahmedabad
event_id =  11  type_id=  1 event_name=  Children Of New Worlds price=  337 event_date=  2021-03-03 location=  Ahmedabad event_time=  10:00:00 dura
tion=  1 total_tickets=  9 tickets_left=  8 rating=  1
event_id =  21  type_id=  1 event_name=  Foreigners And Children price=  368 event_date=  2021-06-10 location=  Ahmedabad event_time=  10:00:00 dur
ation=  3 total_tickets=  9 tickets_left=  8 rating=  1
event_id =  100  type_id=  4 event_name=  Assassins And Children price=  210 event_date=  2021-07-14 location=  Ahmedabad event_time=  10:00:00 dur
ation=  2 total_tickets=  5 tickets_left=  5 rating=  2
event_id =  1  type_id=  1 event_name=  Commander Of Moondust price=  177 event_date=  2021-01-22 location=  Ahmedabad event_time=  06:00:00 durati
on=  3 total_tickets=  8 tickets_left=  6 rating=  1
event_id =  8  type_id=  1 event_name=  Guest Of Our Ship price=  494 event_date=  2021-11-25 location=  Ahmedabad event_time=  11:00:00 duration=
 2 total_tickets=  5 tickets_left=  4 rating=  5
event_id =  33  type_id=  1 event_name=  Favor Of The Eyes price=  313 event_date=  2021-03-14 location=  Ahmedabad event_time=  01:00:00 duration=
  2 total_tickets=  9 tickets_left=  9 rating=  5
PS C:\Users\Lenovo>
```

4. Show details of the user entered table name.

```
# Query 4
while(1):
    print('--------------------------------------')
    print('Welcome to the Menu...')
    print('1: Managers')
    print('2: Customers')
    print('3: event_type')
    print('4: event_details')
    print('5: events_booked')
    print('6: exit')
    print('--------------------------------------')
    choice = int(input('Enter your choice:'))
    print('--------------------------------------')
    if choice == 1:
        print('Manager details')
        ch="select * from main_db.manager order by m_id"
        cursor.execute(ch)
        rows=cursor.fetchall()
        for r in rows:
            for c in r:
                print(c,end='  ')
            print()
    elif choice == 2:
        print('customers details')
        ch="select * from main_db.customer"
        cursor.execute(ch)
        rows=cursor.fetchall()
        for r in rows:
            for c in r:
                print(c,end='  ')
            print()
    elif choice == 3:
        print('event_types details')
        ch="select * from main_db.event_type"
        cursor.execute(ch)
        rows=cursor.fetchall()
        for r in rows:
            for c in r:
                print(c,end='  ')
            print()
    elif choice == 4:
        print('event_details details')
        ch="select * from main_db.event_details"
```

```
    cursor.execute(ch)
    rows=cursor.fetchall()
    for r in rows:
      for c in r:
        print(c,end='  ')
      print()
  elif choice == 5:
    print('event_booked details')
    ch="select * from main_db.event_booked"
    cursor.execute(ch)
    rows=cursor.fetchall()
    for r in rows:
      for c in r:
        print(c,end='  ')
      print()
  else:
    print('Thank you for visiting...')
    print()
    break
```

```
--------------------------------------
Welcome to the Menu...
1: Managers
2: Customers
3: event_type
4: event_details
5: events_booked
6: exit
--------------------------------------
Enter your choice:3
--------------------------------------
event_types details
1  1  movies  40
2  2  sports  20
3  3  concerts  10
4  4  audio_launch  30
--------------------------------------
Welcome to the Menu...
1: Managers
2: Customers
3: event_type
4: event_details
5: events_booked
6: exit
--------------------------------------
Enter your choice:6
--------------------------------------
Thank you for visiting...

PS C:\Users\Lenovo>
```