EVENT MANAGEMENT SYSTEM

PROJECT REPORT

By

ROHITH.N (RA23111027010034)

Under the guidance of

Dr. P Rajasekar

in partial fulfilment for the Course
of

21CSC205P-Database Management Systems
Department of Data Science and Business systems

FACULTY OF ENGINEERING AND TECHNOLOGY
SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that 21CSP302L - Project report titled **"EVENT MANAGEMENT SYSTEM"** is the Bonafide work of **"ROHITH.N[RA2311027010034]"** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Dr. Rajasekar	**DR. Kavitha**
ASSISTANT PROFESSOR	PROFESSOR&HEAD
DATA SCIENCE AND BUSINESS SYSTEMS	DATA SCIENCE AND BUSINESS SYSTEMS

ABSTRACT

The purpose of **Event Management System** is to automate the existing manual system by the help of computerized equipment and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. The required software and hardware are easily available and easy to work with.

Event Management System, as described above, can lead to error free, secure, reliable and fast management systems. It can assist the user to coelenterate on their other activities rather than concentrating on the record keeping. Thus, it will help organizations in better utilization of resources. The organization can maintain computerized records without redundant entries. That means that one need not be distracted by information that is not relevant, while being able to reach the information.

The Event Management System is a structured platform developed to streamline the organization, planning, and execution of various events. It centralizes information related to events, venues, and participants, ensuring efficient data handling and reduced redundancy through proper normalization techniques. The system is built using MySQL for backend database management and a basic front-end interface using HTML, CSS, and JavaScript for user interaction. This project demonstrates how a simple yet powerful database-driven application can assist institutions or organizations in managing multiple events effectively, ensuring accuracy, consistency, and ease of access to data.

The aim is to automate its existing manual system by the help of computerized equipment and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. Basically the project describes how to manage for good performance and better services for the clients.

PROBLEM STATEMENT

Organizing events can be a challenging and time-consuming task, especially when managing a large number of attendees, speakers, sponsors, and vendors. Event planners often struggle with coordinating multiple aspects of an event such as scheduling, ticketing, logistics, and communication.

Therefore, there is a need for an efficient and comprehensive event management system that can streamline the entire process of organizing events. Such a system should be able to handle all aspects of an event, from initial planning to post-event analysis. It should be user-friendly, customizable, and able to integrate with other tools and software.

The system should also provide real-time data on attendee engagement, event performance, and revenue generation, enabling event planners to make data-driven decisions and continuously improve their events.

Overall, an effective event management system should simplify the event planning process, save time and resources, and enhance the attendee experience, making it a valuable investment for any organization that hosts events.

- **Manual Registration & Booking** –spreadsheet registrations, leading to errors and inefficiencies.
- **Poor Data Management –** Tracking attendees, sponsors, and event details manually results in data loss or duplication.
- **Difficulty in Event Scheduling** – Managing multiple events, venues, and time slots manually can cause clashes and mismanagement.
- **Lack of Automated Notifications** – Participants often miss important updates due to a lack of automated reminders.
- **Payment & Ticketing Issues** – Handling payments and ticket generation manually can lead to fraud or mismanagement.

Chapter No 1

Problem Understanding:

Manual Event Handling is Inefficient:

Traditional methods of managing events using paper-based records or basic spreadsheets are time-consuming and error-prone. It becomes difficult to track registrations, venue availability, or participant data efficiently.

Difficulty in Managing Multiple Events:

Institutions or organizations that host multiple events simultaneously struggle to coordinate schedules, resources, and teams without a centralized platform. This leads to confusion, overlap in scheduling, and inefficient resource allocation.

Lack of Real-time Updates:

In existing manual systems, real-time updates like changes in schedule, venue updates, or participant cancellations are hard to communicate instantly. This leads to poor communication and event mismanagement.

Duplication and Redundancy of Data:

Participant information is often entered multiple times for different events, which causes redundancy and wastes storage. There is no centralized database to check if a participant is already registered or if venue capacity is reached.

Limited Accessibility and User Interaction:

Stakeholders such as organizers, volunteers, and participants have no single platform where they can interact, check schedules, or track event updates. This reduces engagement and the overall experience of an event.

Venue Management Issues:

Keeping track of available venues, their capacities, and bookings

manually increases the risk of double-booking or misallocating space. This is a major issue when large numbers of events are planned within the same time frame.

Low Data Security and Backup:

Manual record-keeping poses high risks of data loss or unauthorized access. Sensitive participant data (emails, phone numbers) must be protected with access control, which is missing in basic systems.

No Analytics or Report Generation:

Organizers lack the ability to generate useful reports such as event participation statistics, venue utilization, or feedback trends due to the absence of an integrated system that captures and processes such data.

Scalability Limitations:

As the number of events and participants grow, managing them without a systematic backend or database becomes unscalable. Institutions planning to expand their events face this bottleneck quickly.

No Integrated Communication:

Without built-in email or notification systems, informing participants or volunteers becomes challenging. Manual communication is inefficient, especially in large-scale events.

Identification of Entities & Relationships:

Event – Represents various events such as workshops, fests, and seminars held in the institution or organization.

Organizer – The person or group responsible for organizing and managing the events, including scheduling and communication.

Venue – The physical location or platform where the event takes place, including details like capacity and location name.

Participant – Individuals who register or attend events; can be students, guests, or external members.

Registration – Represents the linking data between participants and events, capturing who attends which event and when.

Schedule – Maintains specific event timing and sequences, including start and end time, session names, and breaks.

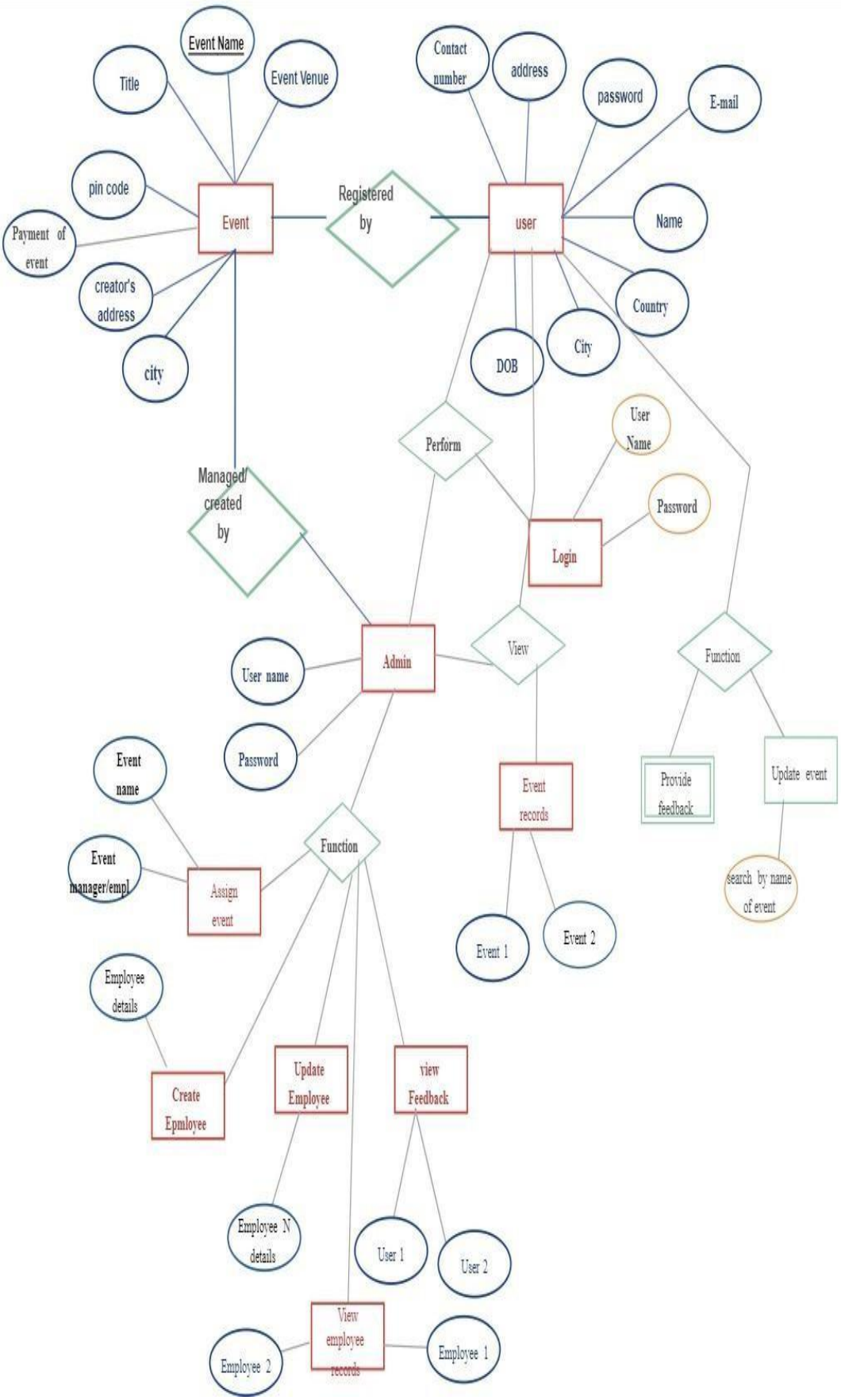
Feedback – Stores post-event feedback given by participants to rate events and provide suggestions or comments.

Certificate – Records certificate generation details for participants after event completion.

Admin/Staff – Manages the backend system, user permissions, event approval, and updates to the platform.

Sponsors – Optional entity representing sponsors providing support or funds for the events.

Construction of DB using ER Model for the project



CHAPTER 2

DESIGN OF RELATIONAL SCHEMAS

Event Table:

Attribute Name	Description	Type
event_id	Unique ID for each event	int
event_name	Name of the event	varchar
event_type	Type of event (Tech/Cultural)	varchar
description	Description of the event	text
organizer_id	ID of the organizer	int
venue_id	ID of the venue	int
schedule_id	ID of the schedule	int

Participant Table:

Attribute Name	Description	Type
participant_id	Unique ID for participant	int
participant_name	Name of participant	varchar
email	Email address	varchar
phone	Contact number	varchar
college_name	Name of the college	varchar

Registration Table:

Attribute Name	Description	Type
registration_id	Unique ID for registration	int
event_id	ID of the event	int
participant_id	ID of the participant	int
registration_date	Date of registration	date

📍 Venue Table:

Attribute Name	Description	Type
venue_id	Unique ID for venue	int
venue_name	Name of the venue	varchar
location	Venue location	varchar
capacity	Total capacity	int

⏰ Schedule Table:

Attribute Name	Description	Type
schedule_id	Unique schedule ID	int
event_date	Date of event	date
start_time	Start time	time
end_time	End time	time

🗣️ Feedback Table:

Attribute Name	Description	Type
feedback_id	Unique feedback ID	int
participant_id	ID of participant	int
event_id	ID of event	int
rating	Rating out of 5	int
comments	Feedback comments	text

Creation of Database And tables:

```
mysql> CREATE TABLE User8 (
    ->     user_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     name VARCHAR(100),
    ->     email VARCHAR(100) UNIQUE,
    ->     phone VARCHAR(15),
    ->     role ENUM('Attendee', 'Organizer')
    -> );
mysql> CREATE TABLE Venue8 (
    ->     venue_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     venue_name VARCHAR(100),
    ->     location VARCHAR(100),
    ->     capacity INT
    -> );
mysql> CREATE TABLE Event8 (
    ->     event_id INT PRIMARY KEY,
    ->     event_name VARCHAR(100),
    ->     event_date DATE,
    ->     venue_id INT,
    ->     organizer_id INT,
    ->     FOREIGN KEY (venue_id) REFERENCES Venue(venue_id),
    ->     FOREIGN KEY (organizer_id) REFERENCES Organizer(organizer_id)
    -> );
mysql> CREATE TABLE Registration8 (
    ->     reg_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     user_id INT,
    ->     event_id INT,
    ->     status ENUM('Pending', 'Confirmed'),
    ->     FOREIGN KEY (user_id) REFERENCES User(user_id),
```

```
    ->     FOREIGN KEY (event_id) REFERENCES Event(event_id)
    -> );
mysql> CREATE TABLE Payment8 (
    ->     payment_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     user_id INT,
    ->     event_id INT,
    ->     amount DECIMAL(10,2),
    ->     status ENUM('Paid', 'Pending'),
    ->     FOREIGN KEY (user_id) REFERENCES User(user_id),
    ->     FOREIGN KEY (event_id) REFERENCES Event(event_id)
    -> );
mysql> CREATE TABLE Feedback8 (
    ->     feedback_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     user_id INT,
    ->     event_id INT,
    ->     rating INT CHECK (rating BETWEEN 1 AND 5),
    ->     comments TEXT,
    ->     FOREIGN KEY (user_id) REFERENCES User(user_id),
    ->     FOREIGN KEY (event_id) REFERENCES Event(event_id)
    -> );
mysql> INSERT INTO User8 (name, email, phone, role) VALUES
    -> ('Rohith', 'rohith@example.com', '9876543210', 'Attendee'),
    -> ('Nikilesh', 'nikilesh@example.com', '9876543211', 'Organizer');
mysql> INSERT INTO Venue8 (venue_name, location, capacity) VALUES
    -> ('Auditorium A', 'Main Block', 500),
    -> ('Hall B', 'Tech Park', 200);
mysql> INSERT INTO Registration8 (user_id, event_id, status) VALUES
    -> (1, 1, 'Confirmed'),
    -> (1, 2, 'Pending');
```

```
mysql> INSERT INTO Payment8 (user_id, event_id, amount, status) VALUES
    -> (1, 1, 200.00, 'Paid'),
    -> (1, 2, 150.00, 'Pending');
mysql> INSERT INTO Payment8 (user_id, event_id, amount, status) VALUES
    -> (1, 1, 200.00, 'Paid'),
    -> (1, 2, 150.00, 'Pending');
mysql> INSERT INTO Feedback8 (user_id, event_id, rating, commen ts)
VALUES
    -> (1, 1, 5, 'Amazing Event!'),
    -> (1, 2, 4, 'Well organized.');
```

```
mysql> SELECT * FROM User8;
+---------+----------+--------------------+------------+-----------+
| user_id | name     | email              | phone      | role      |
+---------+----------+--------------------+------------+-----------+
|       1 | Rohith   | rohith@example.com | 9876543210 | Attendee  |
|       2 | Nikilesh | nikilesh@example.com | 9876543211 | Organizer |
+---------+----------+--------------------+------------+-----------+
2 rows in set (0.00 sec)
```

```
mysql> DESC Event8;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| event_id    | int         | NO   | PRI | NULL    |       |
| event_name  | varchar(100) | YES  |     | NULL    |       |
| event_date  | date        | YES  |     | NULL    |       |
| venue_id    | int         | YES  | MUL | NULL    |       |
| organizer_id | int        | YES  | MUL | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.02 sec)
```

```
mysql> SELECT U.name, E.event_name, R.status
    -> FROM Registration8 R
    -> JOIN User U ON R.user_id = U.user_id
    -> JOIN Event E ON R.event_id = E.event_id;
+--------+------------------+-----------+
| name   | event_name       | status    |
+--------+------------------+-----------+
| Rohith | Coding Hackathon | Confirmed |
| Rohith | Tech Talk        | Pending   |
+--------+------------------+-----------+
```

CHAPTER 3

Complex queries

```
mysql> CREATE TABLE Users (
    ->     UserID INT PRIMARY KEY AUTO_INCREMENT,
    ->     Name VARCHAR(100),
    ->     Role VARCHAR(50),
    ->     Email VARCHAR(100),
    ->     Password VARCHAR(50)
    -> );
mysql> INSERT INTO Users (Name, Role, Email, Password)
    -> VALUES ('Ravi Kumar', 'Organizer', 'ravi@example.com', 'pass123');
mysql> CREATE TABLE Venues (
    ->     VenueID INT PRIMARY KEY AUTO_INCREMENT,
    ->     Name VARCHAR(100),
    ->     Location VARCHAR(100),
    ->     Capacity INT
    -> );
mysql> INSERT INTO Venues (Name, Location, Capacity)
    -> VALUES ('Auditorium', 'Main Block', 300);
mysql> CREATE TABLE Events (
    ->     EventID INT PRIMARY KEY AUTO_INCREMENT,
    ->     Name VARCHAR(100),
    ->     OrganizerID INT,
    ->     Type VARCHAR(50),
    ->     Date DATE,
    ->     VenueID INT,
    ->     FOREIGN KEY (OrganizerID) REFERENCES Users(UserID),
    ->     FOREIGN KEY (VenueID) REFERENCES Venues(VenueID)
```

```
    -> );
mysql> INSERT INTO Events (Name, OrganizerID, Type, Date, VenueID)
    -> VALUES ('Tech Fest 2025', 1, 'Technical', '2025-05-10', 1);
mysql> CREATE TABLE Participants (
    ->     ParticipantID INT PRIMARY KEY AUTO_INCREMENT,
    ->     Name VARCHAR(100),
    ->     Email VARCHAR(100),
    ->     Phone VARCHAR(15),
    ->     EventID INT,
    ->     FOREIGN KEY (EventID) REFERENCES Events(EventID)
    -> ); mysql> INSERT INTO Participants (Name, Email, Phone, EventID)
    -> VALUES ('Anjali', 'anjali@example.com', '9999999999', 1);
mysql> CREATE TABLE Payment_Log (
    ->     LogID INT PRIMARY KEY AUTO_INCREMENT,
    ->     ParticipantID INT,
    ->     Amount DECIMAL(10,2),
    ->     PaymentDate DATETIME,
    ->     LoggedAt DATETIME DEFAULT CURRENT_TIMESTAMP
    -> );
mysql> DELIMITER //
mysql> CREATE TRIGGER after_payment_insert
    -> AFTER INSERT ON Payments
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO Payment_Log (ParticipantID, Amount, PaymentDate)
    ->     VALUES (NEW.ParticipantID, NEW.Amount, NEW.Date);
    -> END;
    -> //
mysql> CREATE TABLE Payments (
```

```
    ->     PaymentID INT PRIMARY KEY AUTO_INCREMENT,
    ->     ParticipantID INT,
    ->     Amount DECIMAL(10,2),
    ->     Status VARCHAR(20),
    ->     Date DATETIME,
    ->     FOREIGN KEY (ParticipantID) REFERENCES Participants(ParticipantID)
    -> );
mysql> INSERT INTO Payments (ParticipantID, Amount, Status, Date)
    -> VALUES (1, 500.00, 'Completed', NOW());
mysql> SELECT
    ->     p.ParticipantID,
    ->     p.Name AS ParticipantName,
    ->     e.Name AS EventName,
    ->     e.Date,
    ->     v.Name AS VenueName
    -> FROM
    ->     Participants p
    -> JOIN
    ->     Events e ON p.EventID = e.EventID
    -> JOIN
    ->     Venues v ON e.VenueID = v.VenueID
    -> JOIN
    ->     Payments pay ON p.ParticipantID = pay.ParticipantI
    -> WHERE
    ->     pay.Status = 'Completed';
```

ParticipantID	ParticipantName	EventName	Date	VenueName
1	Anjali	Tech Fest 2025	2025-05-10	Auditorium

```
mysql> SELECT * FROM Users;
+--------+------------+----------+------------------+----------+
| UserID | Name       | Role     | Email            | Password |
+--------+------------+----------+------------------+----------+
|      1 | Ravi Kumar | Organizer | ravi@example.com | pass123  |
|      2 | Ravi Kumar | Organizer | ravi@example.com | pass123  |
+--------+------------+----------+------------------+----------+
```

```
mysql> SELECT * FROM Venues;
+---------+------------+------------+----------+
| VenueID | Name       | Location   | Capacity |
+---------+------------+------------+----------+
|       1 | Auditorium | Main Block |      300 |
|       2 | Auditorium | Main Block |      300 |
+---------+------------+------------+----------+
```

```
mysql> SELECT * FROM Events;
+---------+---------------+-------------+----------+------------+---------+
| EventID | Name          | OrganizerID | Type     | Date       | VenueID |
+---------+---------------+-------------+----------+------------+---------+
|       1 | Tech Fest 2025 |           1 | Technical | 2025-05-10 |       1 |
|       2 | Tech Fest 2025 |           1 | Technical | 2025-05-10 |       1 |
+---------+---------------+-------------+----------+------------+---------+
```

```
mysql> SELECT * FROM Participants;
+---------------+--------+--------------------+------------+---------+
| ParticipantID | Name   | Email              | Phone      | EventID |
+---------------+--------+--------------------+------------+---------+
|             1 | Anjali | anjali@example.com | 9999999999 |       1 |
|             2 | Rohith | rohith@example.com | 8888888888 |       1 |
|             3 | Anjali | anjali@example.com | 9999999999 |       1 |
+---------------+--------+--------------------+------------+---------+
```

```
mysql> SELECT * FROM Payments;
+-----------+---------------+--------+-----------+---------------------+
| PaymentID | ParticipantID | Amount | Status    | Date                |
+-----------+---------------+--------+-----------+---------------------+
|         1 |             1 | 500.00 | Completed | 2025-04-05 00:08:14 |
|         2 |             2 | 500.00 | Completed | 2025-04-08 07:19:42 |
|         3 |             2 | 500.00 | Completed | 2025-04-08 08:44:09 |
|         4 |             1 | 500.00 | Completed | 2025-04-08 18:30:14 |
+-----------+---------------+--------+-----------+---------------------+
```

```
mysql> SELECT * FROM Payment_Log;
+-------+---------------+--------+---------------------+---------------------+
| LogID | ParticipantID | Amount | PaymentDate         | LoggedAt            |
+-------+---------------+--------+---------------------+---------------------+
|     1 |             2 | 500.00 | 2025-04-08 07:19:42 | 2025-04-08 07:19:42 |
|     2 |             2 | 500.00 | 2025-04-08 08:44:09 | 2025-04-08 08:44:09 |
|     3 |             1 | 500.00 | 2025-04-08 18:30:14 | 2025-04-08 18:30:14 |
+-------+---------------+--------+---------------------+---------------------+
```

CHAPTER 4

ANALYSING THE PITFALL AND IDENTIFYING DEPENDENCIES

Pitfalls in Tables:

1. **Event Table:**
    - Data redundancy may occur if multiple events are organized by the same organizer.
    - Insertion anomaly: Cannot insert event details without organizer ID.
    - Deletion anomaly: Deleting an event may remove essential organizer info if stored together.
2. **Venue Table:**
    - Repetition of venue location data for each event scheduled at the same venue.
    - Update anomaly: Changing venue address needs multiple updates.
3. **Participant Table:**
    - Redundant storage of participant details if they attend multiple events.
    - Deletion anomaly: Removing a participant from an event may remove personal details.
4. **Organizer Table:**
    - Same organizer might be stored with duplicate phone numbers or names if not properly constrained.
5. **Event Schedule Table:**
    - If event date and time are embedded together, time format inconsistency may arise.
    - Potential redundancy with event ID and venue ID pairing.
6. **Feedback Table:**

Feedback linked to participant and event might lead to transitive dependencies if not normalized.

Dependencies in Tables:

1. **Event Table:**
    - Functional: Event_ID → Event_Name, Event_Date, Organizer_ID
    - Transitive: Event_ID → Organizer_ID → Organizer_Name
2. **Venue Table:**
    - Functional: Venue_ID → Venue_Name, Location, Capacity
3. **Participant Table:**
    - Functional: Participant_ID → Name, Email, Phone
4. **Organizer Table:**
    - Functional: Organizer_ID → Organizer_Name, Contact_Number
5. **Event_Schedule Table:**
    - Composite: Event_ID + Venue_ID → Date, Time
6. **Feedback Table:**
    - Composite: Participant_ID + Event_ID → Rating, Comment

NORMALIZATION

Before Normalization(1NF):

```
mysql> SELECT * FROM Unnormalized_Event;
+---------+--------------+-----------+-------------+-------------------+
| EventID | EventName    | EventDate | Venue       | Participants      |
+---------+--------------+-----------+-------------+-------------------+
|     101 | Tech Talk    | 2025-05-01 | Auditorium A | Akhil, Ravi, Sneha |
|     102 | Hackathon    | 2025-05-02 | Lab 1       | Meena, Kiran      |
|     103 | Cultural Fest | 2025-05-03 | Open Ground | Hari, Divya, Manoj |
+---------+--------------+-----------+-------------+-------------------+
```

After Normalization:

mysql> SELECT * FROM Event_1NF;

EventID	EventName	OrganizerName	ParticipantName	Location
101	Tech Talk	Rohith	Akhil	Auditorium A
101	Tech Talk	Rohith	Ravi	Auditorium A
101	Tech Talk	Rohith	Sneha	Auditorium A
102	Hackathon	Jahnavi	Meena	Lab 1
102	Hackathon	Jahnavi	Kiran	Lab 1
103	Cultural Fest	Karthik	Hari	Open Ground
103	Cultural Fest	Karthik	Divya	Open Ground
103	Cultural Fest	Karthik	Manoj	Open Ground
101	Tech Talk	Rohith	Akhil	Auditorium A
101	Tech Talk	Rohith	Ravi	Auditorium A
101	Tech Talk	Rohith	Sneha	Auditorium A
102	Hackathon	Jahnavi	Meena	Lab 1
102	Hackathon	Jahnavi	Kiran	Lab 1
103	Cultural Fest	Karthik	Hari	Open Ground
103	Cultural Fest	Karthik	Divya	Open Ground
103	Cultural Fest	Karthik	Manoj	Open Ground
101	Tech Talk	Rohith	Akhil	Auditorium A
101	Tech Talk	Rohith	Ravi	Auditorium A
101	Tech Talk	Rohith	Sneha	Auditorium A
102	Hackathon	Jahnavi	Meena	Lab 1
102	Hackathon	Jahnavi	Kiran	Lab 1
103	Cultural Fest	Karthik	Hari	Open Ground
103	Cultural Fest	Karthik	Divya	Open Ground
103	Cultural Fest	Karthik	Manoj	Open Ground
1	Tech Fest 2025	2025-05-10	Auditorium	Anjali
1	Tech Fest 2025	2025-05-10	Auditorium	Rohith
1	Tech Fest 2025	2025-05-10	Auditorium	Nikilesh

Before Normalization(2NF):

mysql> SELECT * FROM Event_Before2NF;

EventID	EventName	EventDate	ParticipantName	Venue
101	Tech Talk	2025-05-01	Akhil	Auditorium A
101	Tech Talk	2025-05-01	Ravi	Auditorium A
101	Tech Talk	2025-05-01	Sneha	Auditorium A
102	Hackathon	2025-05-02	Meena	Lab 1
102	Hackathon	2025-05-02	Kiran	Lab 1
103	Cultural Fest	2025-05-03	Hari	Open Ground
103	Cultural Fest	2025-05-03	Divya	Open Ground
103	Cultural Fest	2025-05-03	Manoj	Open Ground

After Normalization:

mysql> SELECT * FROM Events_2NF;

EventID	Name	Date	Venue
1	Tech Fest 2025	2025-05-10	Auditorium
2	Tech Fest 2026	2025-05-10	Tech Park

mysql> SELECT * FROM Participants_2NF;

ParticipantID	Name	Email	Phone	EventID
1	Anjali	anjali@example.com	9999999999	1
2	Rohith	rohith@example.com	8888888888	1
3	Nikilesh	nikilesh@example.com	76809876780	2

Before Normalization(3NF):

```
mysql> SELECT * FROM Event_Before3NF;
+---------+---------------+-----------+-----------------+--------------+---------------+----------+
| EventID | EventName     | EventDate | ParticipantName | VenueName    | Location      | Capacity |
+---------+---------------+-----------+-----------------+--------------+---------------+----------+
|     101 | Tech Talk     | 2025-05-01 | Akhil           | Auditorium A | Block A       |      300 |
|     101 | Tech Talk     | 2025-05-01 | Ravi            | Auditorium A | Block A       |      300 |
|     102 | Hackathon     | 2025-05-02 | Meena           | Lab 1        | Block B       |      100 |
|     102 | Hackathon     | 2025-05-02 | Kiran           | Lab 1        | Block B       |      100 |
|     103 | Cultural Fest | 2025-05-03 | Divya           | Open Ground  | Campus Center |     1000 |
+---------+---------------+-----------+-----------------+--------------+---------------+----------+
```

After Normalization:

```
+---------+-----------------+------------+-------------+-------------+----------+
| EventID | EventName       | EventDate  | VenueName   | Location    | Capacity |
+---------+-----------------+------------+-------------+-------------+----------+
|       1 | Tech Fest 2025 | 2025-05-10 | Auditorium  | Main Block  |      300 |
|       2 | Tech Fest 2026 | 2025-05-10 | Tech Park   | Main Block  |      500 |
+---------+-----------------+------------+-------------+-------------+----------+
```

CHAPTER 5

IMPLEMENTATION OF CONCURRENCY CONTROL AND RECOVERY MECHANISMS

1. Concurrency Control:

Concurrency control ensures correctness and consistency when multiple transactions execute simultaneously.

1.1. Lock-Based Protocols:

- **Binary locks** (locked/unlocked)
- **Shared (S) & Exclusive (X) locks** – Used in **Two-Phase Locking (2PL)**
- **Strict 2PL**: Holds all locks until transaction ends.

1.2. Timestamp-Based Protocols:

- Assigns timestamps to transactions.
- Ensures **serializability** based on timestamps.

1.3. Optimistic Concurrency Control (OCC):

- Transactions proceed without locks.

- Checks for conflicts at the commit time.

1.4. Multi version Concurrency Control (MVCC):

- Keeps multiple versions of data.
- Readers don't block writers (used in PostgreSQL, Oracle).

2. Recovery Mechanisms:

Recovery mechanisms restore the database to a consistent state after a crash or failure.

2.1 Log-Based Recovery:

- Maintains a **log file** (Write-Ahead Logging - WAL).
- Before/After images are stored.
- **Undo** & **Redo** operations.

2.2 Checkpointing:

- Periodically saves the DB state.
- Reduces recovery time.

2.3 Shadow Paging:

- Maintains two-page tables (current and shadow).
- No need to undo changes — just revert to the shadow table.

2.4 ARIES (Advanced Recovery):

- **Analysis → Redo → Undo** phases.
- Used in many modern DBMS systems.

```
<!DOCTYPE html>
<html>
<head>
  <title>Concurrency & Recovery Demo</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    .log { border: 1px solid #ccc; padding: 10px; height: 200px; overflow-y: scroll; }
```

```
    button { margin: 5px; }
  </style>
</head>
<body>
  <h2>Concurrency Control & Recovery Simulator</h2>

  <button onclick="startTransaction('T1')">Start Transaction T1</button>
  <button onclick="startTransaction('T2')">Start Transaction T2</button>
  <button onclick="commitTransaction()">Commit</button>
  <button onclick="crash()">Simulate Crash</button>
  <button onclick="recover()">Recover</button>

  <h3>Transaction Log:</h3>
  <div class="log" id="log"></div>

  <script>
    let log = [];
    let lockTable = { 'data': null };
    let currentTransaction = null;

    function logAction(action) {
      log.push(action);
      document.getElementById("log").innerHTML = log.join("<br>");
    }

    function startTransaction(tid) {
      if (lockTable.data && lockTable.data !== tid) {
        logAction(` ${tid} is waiting for lock (held by ${lockTable.data})`);
      } else {
        lockTable.data = tid;
        currentTransaction = tid;
        logAction(`☑ ${tid} started and acquired lock on data`);
      }
    }

    function commitTransaction() {
      if (currentTransaction) {
        logAction(` ${currentTransaction}  committed`);
        lockTable.data = null;
        currentTransaction = null;
      } else {
        logAction("⚠ No active transaction to commit");
      }
```

```
    }

    function crash() {
      logAction(" System crashed!");
      currentTransaction = null;
    }

    function recover() {
      const lastLog = log[log.length - 1];
      if (lastLog && lastLog.includes("committed")) {
        logAction(" Recovery: Last transaction committed successfully. No rollback needed.");
      } else {
        logAction(" Recovery: Last transaction did not commit. Rolling back...");
      }
      lockTable.data = null;
    }
  </script>
</body>
</html>
```

SCREENSHOT:

CHAPTER 6
CODE FOR PROJECT

- <!DOCTYPE html>
- <html lang="en">
- <head>
- <meta charset="UTF-8" />
- <meta name="viewport" content="width=device-width, initial-scale=1.0" />
- <title>Event Management System</title>
- <style>
- body {
- font-family: Arial, sans-serif;
- background: #f4f4f4;
- margin: 0;
- padding: 0;
- }
- header {
- background: #007bff;
- color: white;
- padding: 1rem;
- text-align: center;
- }

- nav {
- background: #333;
- padding: 0.5rem;
- display: flex;
- justify-content: center;
- }
- nav a {
- color: white;
- text-decoration: none;
- margin: 0 1rem;
- }
- nav a:hover {
- text-decoration: underline;
- }
- .container {
- padding: 2rem;
- }
- .hidden {
- display: none;
- }
- form input, form button {
- display: block;
- margin-bottom: 1rem;
- padding: 0.5rem;
- width: 100%;
- max-width: 400px;
- }
- table {
- width: 100%;
- border-collapse: collapse;
- }
- table, th, td {
- border: 1px solid #ccc;
- }
- th, td {
- padding: 0.75rem;
- text-align: left;
- }
- </style>
- </head>
- <body>

- <header>
- <h1>Event Management System</h1>
- </header>

- <nav>
- Home
- Add Event
- View Events
- </nav>

-
- <section id="home">
- <h2>Welcome to Event Management System</h2>
- This platform helps you manage events and participants easily using a simple interface.
- </section>

- <section id="addEvent" class="hidden">
- <h2>Add Event</h2>
- <form id="eventForm">
- <input type="text" id="eventName" placeholder="Event Name" required />
- <input type="text" id="eventDate" placeholder="Event Date (YYYY-MM-DD)" required />
- <input type="text" id="venue" placeholder="Venue" required />
- <button type="submit">Add Event</button>
- </form>
- </section>

- <section id="viewEvents" class="hidden">
- <h2>View Events</h2>
- <table>
- <thead>
- <tr>
    - <th>Event Name</th>
    - <th>Date</th>
    - <th>Venue</th>
- </tr>
- </thead>

- <tbody id="eventsTable">
- <!-- JS will populate this -->
- </tbody>
- </table>
- </section>
-

- <script>
- const events = [];

- function showSection(sectionId) {
- document.querySelectorAll('section').forEach(section => {
- section.classList.add('hidden');
- });
- document.getElementById(sectionId).classList.remove('hidden');
- }

- document.getElementById('eventForm').addEventListener('submit', function(e) {
- e.preventDefault();
- const name = document.getElementById('eventName').value;
- const date = document.getElementById('eventDate').value;
- const venue = document.getElementById('venue').value;
- events.push({ name, date, venue });
- alert('Event added successfully!');
- this.reset();
- renderEvents();
- showSection('viewEvents');
- });

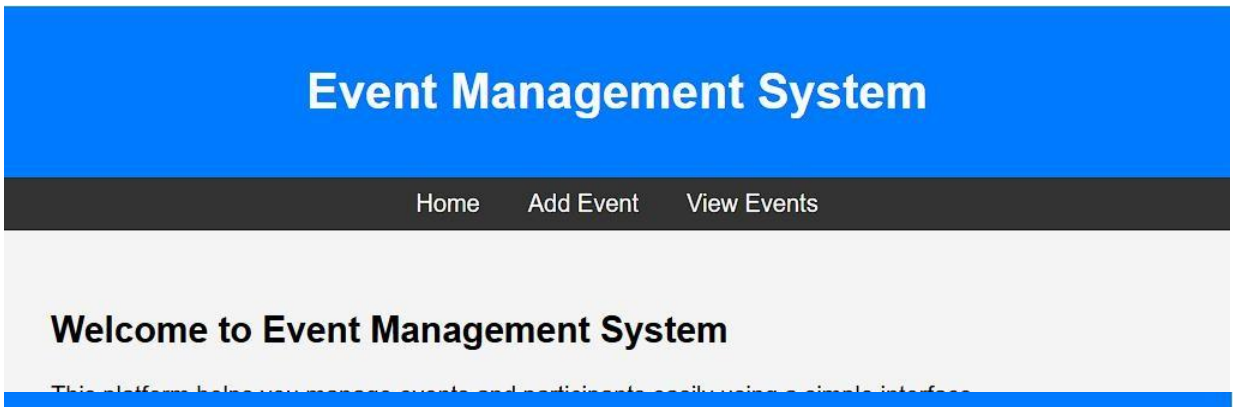
- function renderEvents() {
- const table = document.getElementById('eventsTable');
- table.innerHTML = '';
- events.forEach(event => {
- const row = `<tr><td>${event.name}</td><td>${event.date}</td><td>${event.venue}</td></tr>`;
- table.innerHTML += row;
- });
- }

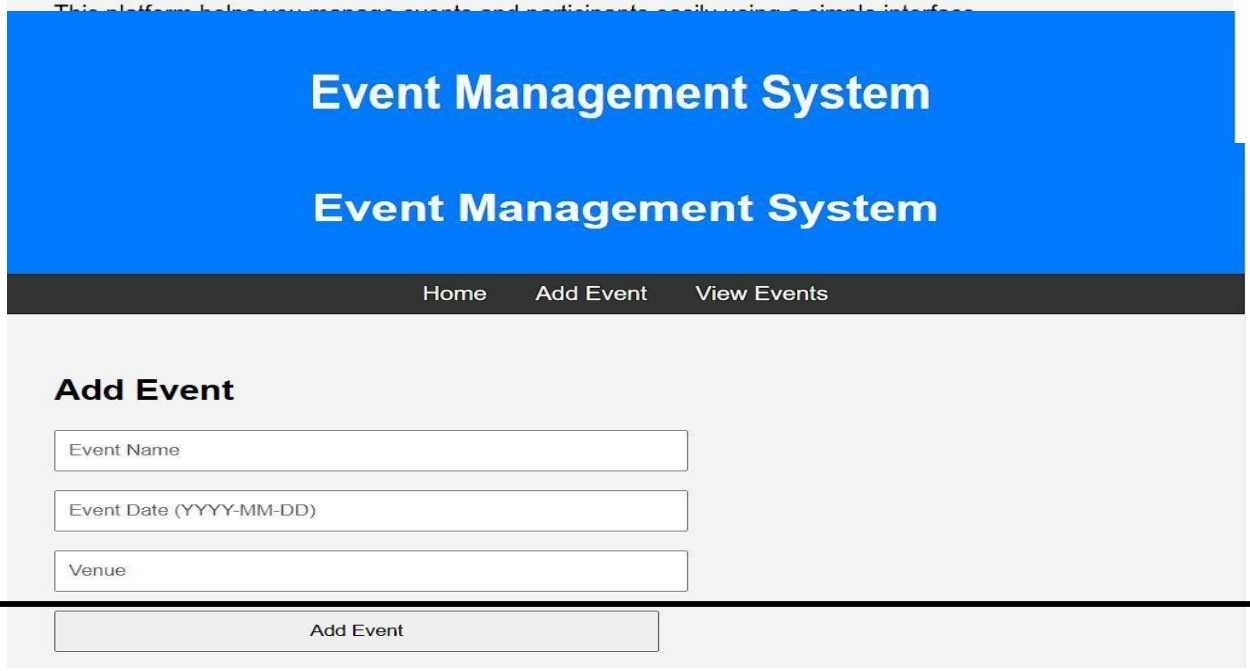
- // Load home initially
- showSection('home');
- </script>
- </body>

</html>

CHAPTER 7

Result and Discussion (Screen shots of the implementation with front end.)





CONCLUSION

The **Event Management System** project involved organizing unstructured data into a clean, efficient format using database normalization up to 3NF. We started with raw data containing redundancy and applied 1NF, 2NF, and 3NF to remove repetition and ensure data integrity. SQL queries were used to create and manage tables for events, participants, and venues. A simple front-end using HTML, CSS, and JavaScript was designed to display and interact with event data. This system helps in smooth event handling, participant tracking, and venue management. Overall, the project ensures better data organization and user accessibility without using complex frameworks.