Event Management System - Project Report

# 1. Introduction

Event Management plays an important role in modern society, where organizing conferences, music shows, workshops, and cultural programs requires systematic planning. Managing events manually often leads to errors such as double bookings, capacity mismanagement, and difficulty tracking attendees.  
  
To overcome these challenges, the Event Management System database is developed using SQL. This system provides an efficient way to store, update, and retrieve information about organizers, venues, events, and attendees. It ensures data integrity through relationships between tables and supports advanced SQL features like constraints, queries, triggers, and stored procedures.

# 2. Objectives

- To maintain event details such as title, description, date, and venue.  
- To store information about organizers and attendees.  
- To manage event registrations efficiently.  
- To implement CRUD operations (Create, Read, Update, Delete).  
- To apply advanced SQL features like aggregate functions, group by, having, subqueries, stored procedures, and triggers.  
- To ensure data consistency and referential integrity using foreign keys and constraints.

# 3. System Requirements

## Software Requirements

- Database: MySQL / SQL Server / Oracle  
- Operating System: Windows/Linux  
- Tools Used: MySQL Workbench, SQL\*Plus, or phpMyAdmin

## Hardware Requirements

- Processor: Intel i3 or higher  
- RAM: 4GB minimum  
- Storage: 10GB free space

# 4. Database Design

The system contains the following tables:  
  
1. Organizer – stores details of event organizers.  
2. Venue – stores venue information like location and capacity.  
3. Event – stores details of events organized.  
4. Attendee – stores information about participants.  
5. Registration – maintains records of event registrations.  
  
Entity–Relationship (E-R) Overview:  
- An Organizer can manage multiple Events.  
- An Event is hosted at one Venue.  
- An Attendee can register for multiple Events.  
- A Registration acts as a link between Event and Attendee.

# 5. SQL Implementation

## 5.1 Create Database and Tables

The following SQL statements create the required database and tables:

CREATE DATABASE EventManagement;  
USE EventManagement;  
  
CREATE TABLE Organizer (  
 OrganizerID INT PRIMARY KEY AUTO\_INCREMENT,  
 Name VARCHAR(100),  
 ContactEmail VARCHAR(100),  
 Phone VARCHAR(15)  
);  
  
CREATE TABLE Venue (  
 VenueID INT PRIMARY KEY AUTO\_INCREMENT,  
 Name VARCHAR(100),  
 Location VARCHAR(200),  
 Capacity INT  
);  
  
CREATE TABLE Event (  
 EventID INT PRIMARY KEY AUTO\_INCREMENT,  
 Title VARCHAR(100),  
 Description TEXT,  
 Date DATE,  
 VenueID INT,  
 OrganizerID INT,  
 TicketPrice DECIMAL(10,2),  
 FOREIGN KEY (VenueID) REFERENCES Venue(VenueID) ON DELETE CASCADE,  
 FOREIGN KEY (OrganizerID) REFERENCES Organizer(OrganizerID) ON DELETE CASCADE  
);  
  
CREATE TABLE Attendee (  
 AttendeeID INT PRIMARY KEY AUTO\_INCREMENT,  
 Name VARCHAR(100),  
 Email VARCHAR(100),  
 Phone VARCHAR(15)  
);  
  
CREATE TABLE Registration (  
 RegistrationID INT PRIMARY KEY AUTO\_INCREMENT,  
 EventID INT,  
 AttendeeID INT,  
 RegistrationDate DATE,  
 FOREIGN KEY (EventID) REFERENCES Event(EventID) ON DELETE CASCADE,  
 FOREIGN KEY (AttendeeID) REFERENCES Attendee(AttendeeID) ON DELETE CASCADE  
);

## 5.2 Insert Sample Records

INSERT INTO Organizer (Name, ContactEmail, Phone) VALUES  
('John Smith', 'john@example.com', '9876543210'),  
('Priya Kumar', 'priya@example.com', '9876500001');  
  
INSERT INTO Venue (Name, Location, Capacity) VALUES  
('City Hall', 'Main Road, Hyderabad', 500),  
('Green Park', 'Sector 7, Delhi', 300);  
  
INSERT INTO Event (Title, Description, Date, VenueID, OrganizerID, TicketPrice) VALUES  
('Tech Conference', 'Annual technology conference', '2025-09-20', 1, 1, 999.99),  
('Music Fest', 'Live music performances', '2025-10-10', 2, 2, 499.50);

## 5.3 Sample Queries

- Update Venue Capacity  
UPDATE Venue SET Capacity = 600 WHERE VenueID = 1;  
  
- Delete an Attendee  
DELETE FROM Attendee WHERE AttendeeID = 2;  
  
- Aggregate Functions  
SELECT COUNT(\*) AS TotalEvents FROM Event;  
SELECT AVG(TicketPrice) AS AveragePrice FROM Event;  
  
- Group By with Having  
SELECT VenueID, COUNT(\*) AS EventCount FROM Event GROUP BY VenueID HAVING EventCount > 1;  
  
- Sub Query  
SELECT \* FROM Event WHERE OrganizerID = (SELECT OrganizerID FROM Organizer WHERE Name = 'Priya Kumar');  
  
- Stored Procedure  
DELIMITER //  
CREATE PROCEDURE GetEventDetails(IN event\_id INT)  
BEGIN  
 SELECT \* FROM Event WHERE EventID = event\_id;  
END //  
DELIMITER ;  
  
- Trigger  
DELIMITER //  
CREATE TRIGGER before\_registration\_insert  
BEFORE INSERT ON Registration  
FOR EACH ROW  
BEGIN  
 DECLARE event\_capacity INT;  
 DECLARE reg\_count INT;  
 SELECT Capacity INTO event\_capacity FROM Venue v  
 JOIN Event e ON v.VenueID = e.VenueID  
 WHERE e.EventID = NEW.EventID;  
  
 SELECT COUNT(\*) INTO reg\_count FROM Registration WHERE EventID = NEW.EventID;  
  
 IF reg\_count >= event\_capacity THEN  
 SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Event capacity full!';  
 END IF;  
END //  
DELIMITER ;

# 6. Results

- Successfully created and managed an Event Management Database.  
- Demonstrated CRUD operations for adding, updating, and deleting records.  
- Applied aggregate functions and advanced SQL features.  
- Implemented stored procedures for reusability and triggers for automatic validation (capacity check).  
- Ensured data consistency with foreign key constraints.

# 7. Conclusion

The Event Management System database provides an efficient way to handle events, organizers, venues, and attendees. It demonstrates the importance of database normalization, relational integrity, and advanced SQL features.  
  
By implementing triggers and stored procedures, the system goes beyond simple storage and provides automation and data security. This project can be extended in the future with a web or mobile application to make event booking and registration user-friendly.