ASSIGNMENT-4

DATABASE MANAGEMENT SYSTEM
CSA0593

SATHISH.R
192324204

Event Management System: Database Design

Description:

The Event Management System (EMS) is designed to manage various aspects of event planning, such as organizing events, handling attendee registrations, managing ticket sales, and ensuring venue availability. The system will support the management of multiple events, track registrations, process payments, and ensure that each event complies with capacity constraints. Additionally, the system will allow for the cancellation of registrations, updating seat availability in real time, and generating reports for event organizers. The main goal is to streamline event management, prevent over-booking, and ensure smooth operations during event planning and execution.

Database Design:

Tables for Events, Venues, Attendees, Registrations, Tickets, and Payments

Events Table: Contains event-specific details such as name, date, and associated venue.

```sql
Sql

CREATE TABLE Events (
    event_id INT PRIMARY KEY,
    event_name VARCHAR(255),
    event_type VARCHAR(100),
    event_date DATE,
    venue_id INT,
    max_occupancy INT,
    FOREIGN KEY (venue_id) REFERENCES Venues(venue_id)
);
```

Venues Table: Holds information about the venue, such as location, name, and seating capacity.

```sql
Sql

CREATE TABLE Venues (
    venue_id INT PRIMARY KEY,
    venue_name VARCHAR(255),
    venue_location VARCHAR(255),
    total_capacity INT
);
```

Attendees Table: Stores data about individuals attending the events, such as contact information.

```sql
Sql

CREATE TABLE Attendees (
    attendee_id INT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(255) UNIQUE,
    phone VARCHAR(15)
);
```

Registrations Table: Links attendees to specific events, tracking who has registered for which event.

```
Sql

CREATE TABLE Registrations (
    registration_id INT PRIMARY KEY,
    attendee_id INT,
    event_id INT,
    registration_date DATE,
    FOREIGN KEY (attendee_id) REFERENCES Attendees(attendee_id),
    FOREIGN KEY (event_id) REFERENCES Events(event_id)
);
```

Tickets Table: Manages the sale of tickets, with details about the ticket, including price and seat number.

```
Sql

CREATE TABLE Tickets (
    ticket_id INT PRIMARY KEY,
    event_id INT,
    ticket_price DECIMAL(10, 2),
    seat_number VARCHAR(10),
    ticket_status VARCHAR(50) DEFAULT 'Available',
    FOREIGN KEY (event_id) REFERENCES Events(event_id)
);
```

Payments Table: Tracks the payment details for tickets purchased, including payment status.

```
sql

CREATE TABLE Payments (
    payment_id INT PRIMARY KEY,
    registration_id INT,
    payment_date DATE,
    payment_amount DECIMAL(10, 2),
    payment_status VARCHAR(50),
    FOREIGN KEY (registration_id) REFERENCES Registrations(registration_id)
);
```

Key Constraints and Features:

Maximum Occupancy Handling:

The **Events Table** includes a `max_occupancy` field, which ensures that no event can exceed its predefined maximum capacity.

When registering attendees or selling tickets, the system will check if the event is already at full capacity.

Event Scheduling Conflicts:

Each event is linked to a venue. The system prevents scheduling multiple events at the same venue on the same date and time by checking for conflicts before allowing event creation.

Stored Procedures:

Event Registration Procedure:

This stored procedure ensures that a new attendee can only register if the event hasn't reached its capacity.

```
sql
CREATE PROCEDURE RegisterForEvent(IN attendee_id INT, IN event_id INT)BEGIN
    DECLARE current_occupancy INT;

    -- Check current occupancy for the event
    SELECT COUNT(*) INTO current_occupancy FROM Registrations WHERE event_id = event_id;

    -- Ensure event hasn't reached maximum occupancy
    IF current_occupancy < (SELECT max_occupancy FROM Events WHERE event_id = event_id) THEN
        INSERT INTO Registrations (attendee_id, event_id, registration_date)
        VALUES (attendee_id, event_id, CURDATE());
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Event is fully booked.';
    END IF;END;
```

Ticket Cancellation Procedure:

Allows attendees to cancel tickets, with seat availability updated automatically.

```
sql
CREATE PROCEDURE CancelTicket(IN ticket_id INT)BEGIN
    -- Mark the ticket as 'Cancelled'
    UPDATE Tickets SET ticket_status = 'Cancelled' WHERE ticket_id = ticket_id;

    -- Update the registration and seat availability
    DELETE FROM Registrations WHERE registration_id = (SELECT registration_id FROM Registrations WHERE
event_id = (SELECT event_id FROM Tickets WHERE ticket_id = ticket_id) AND ticket_id = ticket_id);END;
```

Triggers:

Seat Availability Trigger:

This trigger automatically updates seat availability whenever a new registration is processed, or when a ticket is cancelled.

```
sql
CREATE TRIGGER UpdateSeatAvailability
AFTER INSERT ON RegistrationsFOR EACH ROWBEGIN
    DECLARE available_seats INT;

    -- Check how many available seats are left for the event
    SELECT COUNT(*) INTO available_seats FROM Tickets
    WHERE event_id = NEW.event_id AND ticket_status = 'Available';

    -- If no seats are left, raise an error
    IF available_seats <= 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No available seats left for this event.';
    END IF;END;
```

SQL Queries:

Popular Events Query:

This query lists the top 5 most popular events based on the number of registrations.

```
sql
SELECT e.event_name, COUNT(r.registration_id) AS registrations_countFROM Events eJOIN Registrations r ON
e.event_id = r.event_idGROUP BY e.event_nameORDER BY registrations_count DESC
```

```
LIMIT 5;
```

Revenue by Event Type:

A query to calculate total revenue generated by each event type.

```
sql
SELECT e.event_type, SUM(p.payment_amount) AS total_revenueFROM Events eJOIN Registrations r ON
e.event_id = r.event_idJOIN Payments p ON r.registration_id = p.registration_idGROUP BY e.event_type;
```

Venue Utilization Report:

A query to show how well each venue is being utilized.

```
sql
SELECT v.venue_name, e.event_name, COUNT(r.registration_id) AS attendees_countFROM Venues vJOIN Events e
ON v.venue_id = e.venue_idJOIN Registrations r ON e.event_id = r.event_idGROUP BY v.venue_name,
e.event_name;
```

Conclusion:

The Event Management System provides a robust and scalable solution for managing events, ticket sales, and venue availability. With key features such as registration constraints to avoid overbooking, stored procedures for handling event registrations and ticket cancellations, and triggers for automatic seat updates, the system ensures seamless operations. The ability to generate insightful reports on popular events, revenue, and venue utilization aids event organizers in making data-driven decisions. Overall, this database design streamlines event management, improves attendee experience, and optimizes venue usage.