EVENT MANAGEMENT SYSTEM

By Adarsh

Built with MYSQL

PROJECT OVERVIEW

Goal: Build a relational database to track the full lifecycle of an event platform.

Scope: Managing Users, Event Logistics, Ticket Pricing, and Feedback.

Impact: Transforming raw booking data into actionable business insights.

DATABASE SCHEMA

```sql
CREATE DATABASE event_management;
USE event_management;

CREATE TABLE Users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    city VARCHAR(50),
    age INT
);
```

```sql
CREATE TABLE Events (
    event_id INT PRIMARY KEY AUTO_INCREMENT,
    event_name VARCHAR(100),
    category VARCHAR(50),
    ticket_price DECIMAL(10, 2)
);
```

```sql
CREATE TABLE Registrations (
    registration_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    event_id INT,
    feedback_rating INT,
    attended BOOLEAN,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (event_id) REFERENCES Events(event_id)
);
```

DATA LIFECYCLE MANAGEMENT

Seeding: Populated database with realistic Indian datasets.

```sql
INSERT INTO Users (name, city, age) VALUES
('Aarav Sharma', 'Delhi', 25),
('Ishita Iyer', 'Chennai', 22),
('Vihaan Gupta', 'Mumbai', 28),
('Ananya Singh', 'Bangalore', 24),
('Kabir Malhotra', 'Pune', 30),
('Diya Verma', 'Hyderabad', 21);

INSERT INTO Events (event_name, category, ticket_price) VALUES
('Tech Summit 2024', 'Tech', 1500.00),
('Sunburn Festival', 'Music', 3500.00),
('IPL Screening', 'Sports', 500.00),
('Startup Meetup', 'Tech', 800.00),
('Classical Night', 'Music', 1200.00);

INSERT INTO Registrations (user_id, event_id, feedback_rating, attended) VALUES
(1, 1, 9, 1), (1, 2, 8, 1), (2, 1, 10, 1), (3, 3, 7, 1), (4, 4, 9, 1),
(5, 5, 6, 0), (6, 2, 10, 1), (2, 4, 8, 1), (3, 1, 9, 1), (4, 2, 7, 0);
```

```sql
-- Create (Insert new user)
INSERT INTO Users (name, city, age) VALUES ('Rohan Das', 'Kolkata', 27);

-- Read (Filtering and Sorting)
SELECT * FROM Events WHERE ticket_price > 1000 ORDER BY ticket_price DESC

-- Update
UPDATE Users SET city = 'Jaipur' WHERE user_id = 1;

-- Delete
DELETE FROM Registrations WHERE registration_id = 10;
```

CRUD: Handled Insertions, Updates (User relocations),& Deletions (Cancellations)

ANALYZING DEMAND & SATISFACTION

Q1. Which events drive traffic?

```sql
-- Total registrations and average feedback per event
SELECT e.event_name, COUNT(r.registration_id) AS total_regs, AVG(r.feedback_rating) AS avg_rating
FROM Events e LEFT JOIN Registrations r ON e.event_id = r.event_id
GROUP BY e.event_id;
```

Q2. Gold Standard Events (>8.0)

```sql
-- Events with average rating > 8
SELECT event_name FROM Events e JOIN Registrations r ON e.event_id = r.event_id
GROUP BY e.event_id HAVING AVG(r.feedback_rating) > 8;
```

REVENUE & CATEGORY BENCHMARKING

Q3. Most Profitable Events

```sql
-- Total revenue per event
SELECT event_name, (COUNT(registration_id) * ticket_price) AS revenue
FROM Events e JOIN Registrations r ON e.event_id = r.event_id
GROUP BY e.event_id;
```

Q4. Best in Class (Highest Rating)

```sql
-- Top-performing events by category (Highest Rating)
SELECT category, event_name, MAX(avg_r) FROM
(SELECT category, event_name, AVG(feedback_rating) as avg_r FROM Events JOIN Registrations USING(event_id) GROUP BY event_id) as t
GROUP BY category, event_name;
```

LOYALTY VS. CHURN RISK

Q5. The Power Users

```sql
-- Users who attended more than one event
SELECT name FROM Users u JOIN Registrations r ON u.user_id = r.user_id
WHERE r.attended = 1 GROUP BY u.user_id HAVING COUNT(r.event_id) > 1;
```

Q6. The Cold Leads

```sql
-- Users who have not attended any event
SELECT name FROM Users WHERE user_id NOT IN (SELECT user_id FROM Registrations WHERE attended = 1);
```

PREMIUM USERS & LEADERBOARDS

Q7. High-Ticket Clients

```sql
-- Users who attended the most expensive event
SELECT name FROM Users JOIN Registrations USING(user_id)
WHERE event_id = (SELECT event_id FROM Events ORDER BY ticket_price DESC LIMIT 1) AND attended = 1;
```

Q8. The Power Users

```sql
-- Leaderboard: User name, Total events attended, Average feedback rating
SELECT u.name, COUNT(r.event_id) AS events_attended, AVG(r.feedback_rating) AS avg_given_rating
FROM Users u JOIN Registrations r ON u.user_id = r.user_id
WHERE r.attended = 1
GROUP BY u.user_id
ORDER BY events_attended DESC, avg_given_rating DESC;
```

THE REALITY CHECK (INTENT VS. ACTION)

Q9. Where did we lose them?

```sql
-- Analyze Event popularity
SELECT
    e.event_name,
    e.category,
    COUNT(r.registration_id) AS total_registrations,
    SUM(CASE WHEN r.attended = 1 THEN 1 ELSE 0 END) AS total_attendees
FROM
    Events e
LEFT JOIN
    Registrations r ON e.event_id = r.event_id
GROUP BY
    e.event_id, e.event_name, e.category
ORDER BY
    total_registrations DESC;
```

Q10. Who flakes?

```sql
-- Analyze User Participation
SELECT
    u.name,
    u.city,
    COUNT(r.event_id) AS events_registered,
    SUM(CASE WHEN r.attended = 1 THEN 1 ELSE 0 END) AS events_attended
FROM
    Users u
JOIN
    Registrations r ON u.user_id = r.user_id
GROUP BY
    u.user_id, u.name, u.city
ORDER BY
    events_attended DESC, events_registered DESC;
```

KEY BUSINESS TAKEAWAYS

💰 **Revenue:** Music events drive higher margins than Tech.

📉 **Operations:** High registration doesn't guarantee attendance (Need reminders).

🎯 **Marketing:** Segmentation of "Power Users" vs. "At-Risk" is ready for CRM.

THANK YOU