# THEME PARK MANAGEMENT SYSTEM

Abstract: Theme Park Management System

The Theme Park Management System is a comprehensive software solution designed to streamline the management and operations of a theme park. This project integrates a relational database management system (DBMS) powered by SQL with a user-friendly front-end developed in Java to ensure efficient handling of park activities. It caters to various stakeholders, including administrators, staff, and visitors, by offering tailored functionalities to meet their specific needs.

## Objective

The primary objective of this project is to automate and optimize the management of a theme park by maintaining accurate records, enabling efficient ticketing and reservations, improving visitor experience, and facilitating better decision-making for administrators.

## Key Features

Visitor Management:

Registration and profile management.

Ticket booking (single-day passes, season passes, or event-specific tickets).

Real-time tracking of visitor entry and exit.

Staff Management:

Employee registration, scheduling, and payroll.

Task assignment and performance tracking.

Ride and Attraction Management:

Maintenance schedules for rides.

Real-time ride availability and visitor capacity tracking.

Prioritization of visitor queues (e.g., fast-track tickets).

Event Management:

Scheduling special events, shows, and activities.

Event ticketing and registration management.

Payment and Billing System:

Online and offline payment processing.

Invoice and receipt generation for visitors.

Report Generation:

Daily, weekly, and monthly reports on visitor footfall, revenue, and expenses.

Performance analytics for rides, events, and staff.

Technical Architecture

Database (SQL):

Relational database schema to store data for visitors, employees, tickets, rides, events, and payments.

Query optimization for fast retrieval and updates.

Data integrity constraints for consistency.

Front-End (Java):

Interactive GUI for user interaction using Java Swing/JavaFX.

Role-based access control for administrators, staff, and visitors.

Integration with the database via JDBC.

Backend Logic (Java):

Core functionalities such as ticket allocation, schedule management, and report generation implemented in Java.

Exception handling for seamless operations.

Advantages

Efficiency: Automates repetitive tasks and reduces manual intervention.

Scalability: Can accommodate the growing needs of a larger park.

User Experience: Provides a streamlined and intuitive interface for visitors and staff.

Real-Time Data Access: Offers up-to-date information for effective decision-making.

Potential Use Cases

Small and medium-sized theme parks aiming to digitize their operations.

Large parks seeking to enhance visitor experience and optimize resource utilization.

Event-based amusement parks for seasonal or special-event management.

This project serves as a robust solution for managing theme park operations, enhancing productivity, and delivering a superior visitor experience.

```
-- Visitor Table
CREATE TABLE Visitors (
  VisitorID INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(100),
  Email VARCHAR(100) UNIQUE,
  Phone VARCHAR(15),
  Age INT
);

-- Ticket Table
CREATE TABLE Tickets (
```

```
    TicketID INT PRIMARY KEY AUTO_INCREMENT,
    VisitorID INT,
    TicketType VARCHAR(50), -- e.g., Single Day, Season Pass
    Price DECIMAL(10, 2),
    BookingDate DATE,
    FOREIGN KEY (VisitorID) REFERENCES Visitors(VisitorID)
);
```

JDBC CONNECTIVITY :

```
Import java.sql.Connection;
Import java.sql.DriverManager;
Import java.sql.SQLException;

Public class DBConnection {
    Private static final String URL = “jdbc:mysql://localhost:3306/ThemeParkDB”;
    Private static final String USER = “root”; // Replace with your username
    Private static final String PASSWORD = “password”; // Replace with your password

    Public static Connection getConnection() {
        Try {
            Return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (SQLException e) {
            System.out.println(“Connection failed: “ + e.getMessage());
            Return null;
```

```
    }
  }
}
```

VISITOR REGISTRATION :

```
Import java.sql.Connection;
Import java.sql.PreparedStatement;
Import java.sql.SQLException;

Public class VisitorManager {
  Public static void registerVisitor(String name, String email, String phone, int age) {
    String query = “INSERT INTO Visitors (Name, Email, Phone, Age) VALUES (?, ?, ?, ?)”;
    Try (Connection conn = DBConnection.getConnection();
      PreparedStatement stmt = conn.prepareStatement(query)) {
      Stmt.setString(1, name);
      Stmt.setString(2, email);
      Stmt.setString(3, phone);
      Stmt.setInt(4, age);
      Stmt.executeUpdate();
      System.out.println(“Visitor registered successfully!”);
    } catch (SQLException e) {
      System.out.println(“Error: “ + e.getMessage());
    }
  }
}
```

TICKET BOOKING

```
Import java.sql.Connection;
```

```
Import java.sql.PreparedStatement;

Import java.sql.ResultSet;

Import java.sql.SQLException;

Import java.time.LocalDate;


Public class TicketManager {

   Public static void bookTicket(String email, String ticketType, double price) {

      String getVisitorIdQuery = “SELECT VisitorID FROM Visitors WHERE Email = ?”;

      String insertTicketQuery = “INSERT INTO Tickets (VisitorID, TicketType, Price,
BookingDate) VALUES (?, ?, ?, ?)”;


      Try (Connection conn = DBConnection.getConnection();

         PreparedStatement getVisitorStmt = conn.prepareStatement(getVisitorIdQuery);

         PreparedStatement insertTicketStmt = conn.prepareStatement(insertTicketQuery)) {


        // Get VisitorID using email

        getVisitorStmt.setString(1, email);

        ResultSet rs = getVisitorStmt.executeQuery();


        If (rs.next()) {

           Int visitorId = rs.getInt(“VisitorID”);


           // Insert ticket into Tickets table

           insertTicketStmt.setInt(1, visitorId);

           insertTicketStmt.setString(2, ticketType);

           insertTicketStmt.setDouble(3, price);
```

```
            insertTicketStmt.setDate(4, java.sql.Date.valueOf(LocalDate.now()));
            insertTicketStmt.executeUpdate();

            System.out.println("Ticket booked successfully!");
        } else {
            System.out.println("Visitor not found. Please register first.");
        }
    } catch (SQLException e) {
        System.out.println("Error: " + e.getMessage());
    }
  }
}
```

MAIN CLASS

```
import java.util.Scanner;

public class ThemeParkSystem {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    while (true) {
      System.out.println("\n--- Theme Park Management System ---");
      System.out.println("1. Register Visitor");
      System.out.println("2. Book Ticket");
      System.out.println("3. Exit");
      System.out.print("Enter your choice: ");
      int choice = scanner.nextInt();
```

```
scanner.nextLine(); // Consume newline

switch (choice) {
    case 1:
        System.out.print("Enter Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Email: ");
        String email = scanner.nextLine();
        System.out.print("Enter Phone: ");
        String phone = scanner.nextLine();
        System.out.print("Enter Age: ");
        int age = scanner.nextInt();
        VisitorManager.registerVisitor(name, email, phone, age);
        break;

    case 2:
        System.out.print("Enter Visitor Email: ");
        email = scanner.nextLine();
        System.out.print("Enter Ticket Type (Single Day/Season Pass): ");
        String ticketType = scanner.nextLine();
        System.out.print("Enter Ticket Price: ");
        double price = scanner.nextDouble();
        TicketManager.bookTicket(email, ticketType, price);
        break;

    case 3:
```

```
            System.out.println("Exiting... Goodbye!");
            scanner.close();
            System.exit(0);

        default:
            System.out.println("Invalid choice! Please try again.");
      }
    }
  }
}
```