**Smart Parking Lot Management System**

**Cover Page**

* Project Title: Smart Parking Lot Management System
* Student Name: P.Kanagalakshmi
* Course/Batch: DataAnalysis
* Date:

**Table of Contents**

1. Introduction

2. Objectives

3. System Design

* 3.1 SQL Database Design
* 3.2 Python Integration

4. Implementation

* 4.1SQL Implementation
* 4.2 Python Implementation

5. Sample Queries and Analytics

* 5.1 SQL Queries
* 5.2 Python Analytics

6. Results

7. Conclusion

8. Future Enhancements

## 1.Introduction

The Smart Parking Lot Management System automates parking operations by integrating a MySQL database with Python analytics. It tracks vehicle entries/exits, allocates parking spots, calculates fees, and generates reports to assist administrators.

## 2. Objectives

* Automate parking spot allocation and vehicle tracking.
* Maintain customer and vehicle records.
* Automatically calculate parking fees.
* Provide analytics on revenue, parking duration, and occupancy.
* Generate visual reports using Python.

## 3. System Design

### 3.1 SQL Database Design

**Tables:**

* customers – Stores customer details.
* vehicles – Stores vehicle details linked to customers.
* parking\_spots – Stores spot availability.
* parking\_records – Logs entries/exits and fees.
* parking\_fees – Hourly rates by vehicle type.
* parking\_audit – Logs all insert/update/delete operations.

**Triggers & Procedures:**

* mark\_spot\_occupied – Marks spot occupied on entry.
* mark\_spot\_free – Frees spot on exit.
* audit\_on\_delete – Logs deletions.
* calculate\_fee(rec\_id) – Calculates fee based on vehicle type and duration.

**Views:**

* current\_occupancy – Shows currently occupied spots.

### 3.2 Python Integration

* Python connects to MySQL using PyMySQL.
* Pandas & NumPy: Load and analyze data.
* Matplotlib: Visualize revenue, duration, and trends.

## 4. Implementation

### 4.1 SQL Implementation

* Created database smart\_parking\_lot with tables, triggers, procedures, and sample data.
* Triggers update spot status and maintain audit logs.
* Procedure calculate\_fee computes parking charges.
* Views allow easy access to current occupancy.

4.2 Python Implementation

Example Python tasks:

import pymysql  
import pandas as pd  
import matplotlib.pyplot as plt  
  
connection = pymysql.connect(  
 host="localhost",  
 user="root",  
 password="susi123",  
 database="smart\_parking\_lot"  
)  
  
query = "SELECT \* FROM parking\_records"  
df = pd.read\_sql(query, connection)  
df['duration\_hours'] = (df['exit\_time'] - df['entry\_time']).dt.total\_seconds()/3600  
  
revenue\_by\_type = df.groupby('vehicle\_type')['fee'].sum()  
revenue\_by\_type.plot(kind='bar')  
plt.title("Revenue by Vehicle Type")  
plt.show()

## 5. Sample Queries and Analytics

### 5.1 SQL Queries

* **Current Occupancy:**

SELECT \* FROM current\_occupancy;

* **Longest Parking Sessions:**

SELECT v.plate\_no, TIMESTAMPDIFF(HOUR, pr.entry\_time, pr.exit\_time) AS hours\_parked  
FROM parking\_records pr  
JOIN vehicles v ON pr.vehicle\_id = v.vehicle\_id  
WHERE pr.exit\_time IS NOT NULL  
ORDER BY hours\_parked DESC  
LIMIT 5;

* **Revenue by Vehicle Type:**

SELECT v.vehicle\_type, SUM(pr.fee) AS total\_revenue  
FROM parking\_records pr  
JOIN vehicles v ON pr.vehicle\_id = v.vehicle\_id  
WHERE pr.exit\_time IS NOT NULL  
GROUP BY v.vehicle\_type;

### 5.2 Python Analytics

* Total revenue, average parking duration, and revenue distribution by vehicle type.
* Visualizations provide management insights (bar charts, histograms, line charts).

## 6. Results

* Automated spot allocation and fee calculation.
* Real-time monitoring of parking occupancy.
* Analytical insights into revenue, parking duration, and peak hours.
* Audit trail ensures accountability.

## 7. Future Enhancements

* Mobile app integration for customers.
* Online booking and payment system.
* IoT-enabled real-time parking spot tracking.
* Predictive analytics for peak hours and revenue optimization.

## 8. Conclusion

The project demonstrates the integration of SQL and Python to manage parking efficiently. SQL handles data storage, constraints, triggers, and calculations, while Python provides analytics and visualizations. The system is scalable, reliable, and practical for modern parking management