

TRAVEL MANAGEMENT SYSTEM

DBMS MINI PROJECT REPORT

Submitted by:

MATHAN S

(231801098)

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY IN ARTIFICIAL INTELLIGENCE AND DATA SCIENCE



RAJALAKSHMI ENGINEERING COLLEGE,

ANNA UNIVERSITY, CHENNAI: 602 105

DECEMBER 2024

**RAJALAKSHMI ENGINEERING COLLEGE,
CHENNAI 600 025**

BONAFIDE CERTIFICATE

Certified that this report title “**TRAVEL MANGEMENT SYSTEM**” is the Bonafide work of **MATHAN S(231801098)** who carried out the mini project work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Hod Name with designation
HEAD OF THE DEPARTMENT,
Professor,
Department of AI&DS,
Rajalakshmi Engineering College
Chennai – 602 105.

SIGNATURE

Supervisor name with designation
SUPERVISOR
Assistant Professor,
Department of AI&DS,
Rajalakshmi Engineering College,
Chennai – 602 105.

Submitted for the DBMS Mini project review held on _____

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

Initially I thank the Almighty for being with us through every walk of my life and showering his blessings through the endeavor to put forth this report.

My sincere thanks to our Chairman **Mr. S. MEGANATHAN, M.E., F.I.E.**, and our Chairperson **Dr. (Mrs.)THANGAM MEGANATHAN, M.E., Ph.D.**, for providing me with the requisite infrastructure and sincere endeavoring educating me in their premier institution.

My sincere thanks to **Dr.S.N. MURUGESAN, M.E., Ph.D.**, our beloved Principal for his kind support and facilities provided to complete our work in time.

I express my sincere thanks to **Mr.J.M.Gnanasekar M.E.,Ph.D.**, Head of the Department of Artificial Intelligence and Machine Learning and Artificial Intelligence and Data Science for his guidance and encouragement throughout the project work. I convey my sincere and deepest gratitude to our internal guide, **Mrs.M.Thamizharasi, M.Tech.**, (Ph.d),,Assistant Professor, Department of Artificial Intelligence and Machine Learning, Rajalakshmi Engineering College for his valuable guidance throughout the course of the project.

Finally I express my gratitude to my parents and classmates for their moral support and valuable suggestions during the course of the project.

TABLE OF CONTENTS

S:NO	TITLE	PAGE NO
1.	Introduction	3
1.1	Objectives	3
1.2	Modules	5
2.	Suevey of Technologies	7
2.1	Software Description	7
3.	Requirements and	9
3.1	Analysis	
3.2	Requirements and	9
3.3	Specifications	
3.4	Hardware and Software	9
3.5	Requirements	
	Architecture Diagram	10
	ER Diagram	10
	Normalization	10
4.	Program Code	12
4.1	Customer Functions	12
4.2	Package Functions	12
4.3	Booking Functions	12
4.4	Payment Functions	12
4.5	Main Menu Functions	13
5.	Results and Discussions	14
5.1.	Sample Outputs	14
5.2.	Observations	15
5.3.	Challenges Encounter	15
6.	Database Design	17
6.1.	Entities and Attributes	17
6.2.	Relationships	17
6.3.	ER Diagram	18
6.4.	Representation	18
6.5.	Visual Representation	19
6.6.	Database Schema	20
	Functionality	
7.	Code Implementation	21
7.1.	Sql Source Code	21
7.2.	Output	22
7.3.	Python Source Code	23

7.4.	Output	30
7.5.	Sql Commands	36
8.	Conclusion	38
9.	References	40
9.1.	Python Installation	42
9.2.	Guide	
9.3.	Sql Installation Guide	42
	Configuring Python To	43
	Connect MySQL	
	Installation	

1. Introduction:

The **Travel Management System** is designed as a comprehensive tool to streamline operations for travel agencies by automating and centralizing the management of travel-related data. The system is intended to handle a wide range of functions, from managing customer information to overseeing travel packages, bookings, and payments. By replacing traditional manual processes with automated workflows, the system aims to reduce errors, improve data accuracy, and significantly enhance the speed of data retrieval and processing. This ultimately allows agencies to provide a more efficient and organized service to their clients.

This system addresses several common challenges in the travel industry, such as the high volume of data processing and the need for secure and accurate record-keeping. Agencies often manage complex information across multiple areas, including customer details, package options, booking records, financial transactions, and special offers. The Travel Management System integrates these processes, making it easy for agency staff to access up-to-date information and quickly respond to client inquiries or changes.

Through its user-friendly interface, the system allows travel agents to:

- **Manage Customer Profiles:** Easily store and retrieve personal information, preferences, and travel histories for customers.
- **Oversee Travel Packages:** Create, edit, and remove travel packages with various destinations, accommodations, and pricing options, allowing agencies to offer tailored services.
- **Facilitate Bookings and Reservations:** Quickly book and modify reservations, while tracking the status of each booking to ensure smooth service for clients.
- **Handle Payments:** Securely process and record payments, generate invoices, and track payment histories, reducing the risks associated with financial errors.

By implementing a Travel Management System, agencies can minimize the time spent on routine administrative tasks, enabling staff to focus on enhancing the travel experience for customers. The centralized database structure ensures that all users within the agency have access to accurate, up-to-date information, fostering better coordination and customer satisfaction.

1.1 Objectives:

The **Travel Management System** project is focused on developing a robust, user-friendly database solution tailored to meet the needs of travel agencies. By leveraging an integrated database and application interface, this system is designed to streamline operations, enhance data management, and improve overall efficiency. The primary objectives of the project are as follows:

1. **To Design a Comprehensive Database for Efficient Data Management**

The core of the project is a well-structured relational database that efficiently stores and

organizes all relevant travel data, including customer details, booking information, travel packages, and payment records. This database will support the complex data needs of travel agencies, making it simple to retrieve, update, and manage information without compromising on performance. Properly structured tables, relationships, and indexing are used to optimize data retrieval and storage efficiency.

2. **To Enable CRUD Functionalities for Key Entities through a Python Interface**

CRUD (Create, Read, Update, Delete) operations form the backbone of any data management system. This project provides these essential functionalities across all main data categories:

Customers: The system allows for the creation, viewing, updating, and deletion of customer profiles. This ensures that accurate customer information is available for service personalization and communication.

Packages: CRUD functions allow the addition, modification, and deletion of travel packages, including pricing, destinations, and duration. This feature helps agencies keep their offerings up-to-date.

Bookings: Booking records can be easily created, edited, viewed, or canceled, providing flexibility in managing reservations.

Payments: The system securely processes payments and maintains a detailed transaction history for easy reference, allowing agencies to efficiently manage financial records.

3. **To Establish a Seamless Interface for Database Interaction**

A primary goal of this project is to create an intuitive and user-friendly Python application interface that allows travel agency staff to interact seamlessly with the database. This interface simplifies complex database operations, so users with minimal technical experience can perform a range of tasks without needing direct SQL knowledge. The interface is designed to be straightforward and accessible, helping users complete essential tasks with minimal effort and training.

4. **To Ensure Data Integrity, Minimize Redundancy, and Provide Real-Time Information**

Data integrity is crucial for accurate record-keeping and reliable service delivery. This project aims to enforce integrity through structured data validation, relationship constraints, and the elimination of redundancy. The database is designed to ensure that all data is accurate, consistent, and up-to-date, preventing issues such as duplicate records or outdated information. Real-time updates provide staff with current information on bookings, customer profiles, package availability, and financial transactions, enabling responsive and informed decision-making.

In summary, this project aims to develop a Travel Management System that not only meets the technical needs of travel agencies but also supports their goals of efficiency, accuracy, and

enhanced customer service. By addressing these objectives, the system provides a solid foundation for a modern, automated approach to travel management, ultimately helping agencies improve their operational workflow and customer satisfaction.

- To design a database that efficiently stores and manages customer and booking data.
- To enable CRUD (Create, Read, Update, Delete) functionalities for customers, packages, bookings, and payments through a Python application.
- To establish a seamless interface for interacting with the database, making it easy for users to perform various operations.
- To ensure data integrity, minimize redundancy, and provide accurate real-time information.

1.2 Modules:

The **Travel Management System** consists of several key modules, each designed to handle a specific area of functionality. These modules work together to manage the core aspects of travel agency operations, allowing users to perform necessary actions through a clear, menu-driven interface.

Customer Management

This module focuses on managing all customer-related information, providing a streamlined approach to handle customer records. It allows agency staff to:

Add new customers: Input customer details such as name, contact information, preferences, and travel history.

Update customer records: Modify existing customer information to ensure it remains accurate and up-to-date.

Delete customer records: Remove outdated or unnecessary customer profiles as needed. By organizing customer data efficiently, this module helps the agency offer personalized services and keeps records accurate and easy to access.

Package Management

The Package Management module is dedicated to handling the details of various travel packages offered by the agency. Users can:

Create new packages: Define package details, including destination, itinerary, price, duration, and any special inclusions.

Update package details: Modify existing packages to reflect new offers, discounts, or changes in itinerary.

Delete outdated packages: Remove packages that are no longer available or relevant. This module allows agencies to keep their travel offerings fresh and attractive, helping to meet the changing preferences of their clients.

Booking Management

This module manages the booking process, ensuring that reservations are accurately tracked and modified as needed. Users can:

Insert new bookings: Add reservations for customers, including travel dates, package details, and any additional requirements.

Update existing bookings: Make changes to existing bookings, such as travel dates or package modifications, to accommodate customer requests.

Delete bookings: Cancel reservations that are no longer needed or have been completed. This booking functionality helps the agency handle reservations smoothly, improving efficiency and customer satisfaction by providing up-to-date booking records.

Payment Management

The Payment Management module handles the financial aspects of each booking, ensuring that all payments are recorded accurately. Users can:

Record new payments: Enter payment details, such as the booking ID, amount paid, payment date, and method of payment.

Update payment records: Modify payment information if corrections are needed or additional payments are made.

Delete payment records: Remove payment records associated with canceled bookings or errors. This module ensures that all financial transactions are securely tracked, helping the agency maintain organized and accessible financial records.

System Management

The System Management module serves as the central interface, providing users with a main menu to access each of the above modules. This menu-driven interface enables agency staff to:

Navigate between modules easily: Access any of the system's functionalities from a single starting point.

Perform operations with minimal complexity: Simplify the process of performing CRUD operations on customer, package, booking, and payment records.

Ensure security and efficiency: Restrict access based on user roles (if applicable) and provide a consistent workflow across modules.

This centralized approach not only enhances usability but also provides a seamless experience for agency staff, making it easier to manage the various aspects of travel services. The System Management module is designed to optimize workflows, reduce errors, and provide staff with a straightforward way to handle everyday tasks efficiently.

2. Survey of Technologies

2.1 Software Description:

This project is developed using **Python** for application logic and **MySQL** for database management. Each of these technologies plays a crucial role in ensuring that the Travel Management System is efficient, reliable, and user-friendly. Below is an expanded description of the technologies used:

Python

Python is a versatile, high-level programming language widely recognized for its simplicity and readability, making it a popular choice for both beginners and professionals. In this project, Python serves as the primary language for implementing the application's logic, handling user input, and managing data flow between the user interface and the database. Key benefits and features of Python in this context include:

Ease of Use and Readability: Python's syntax is straightforward, which enhances code readability and maintainability. This makes it ideal for projects that may need to be updated or modified over time.

Extensive Library Support: Python offers a wide range of libraries, such as `mysql-connector` and `SQLAlchemy`, that simplify tasks like database connections, data validation, and error handling. These libraries streamline database interactions, enabling smooth CRUD operations with minimal code complexity.

Cross-Platform Compatibility: Python is compatible with various operating systems, including Windows, macOS, and Linux, allowing for flexibility in deployment. This cross-platform functionality ensures that the system can be easily adapted for different user environments.

Strong Community and Resources: Python's large user community and extensive documentation make it easy to find resources, troubleshoot issues, and incorporate best practices, ensuring a high level of support for development.

In this project, Python is used to create the **menu-driven interface** and to implement the logic for interacting with each module, including Customer Management, Package Management, Booking Management, and Payment Management. By using Python, the system remains user-friendly while maintaining high functionality.

MySQL

MySQL is an open-source relational database management system (RDBMS) known for its speed, reliability, and ease of use. It is widely used in applications where structured, secure, and scalable data storage is essential. In this Travel Management System, MySQL is employed to store, organize, and manage data associated with customers, travel packages, bookings, and payments. Key benefits and features of MySQL in this context include:

Data Integrity and Security: MySQL supports ACID (Atomicity, Consistency, Isolation, Durability) compliance, which ensures that all transactions are processed reliably, preserving

data accuracy and preventing corruption. Its strong data integrity features make it ideal for applications that require secure and consistent data handling.

Efficient Data Storage and Retrieval: MySQL uses a relational structure with tables, rows, and columns, making it highly efficient for storing and retrieving structured data. Indexing and optimized querying enhance performance, enabling quick access to large datasets—a valuable feature for managing a high volume of customer and booking records.

Scalability and Flexibility: MySQL is suitable for small projects but is also highly scalable, allowing for future growth as data volume increases. Its flexibility ensures it can handle various types of queries and data relationships, making it adaptable for more complex applications.

Ease of Integration with Python: MySQL works seamlessly with Python through libraries such as `mysql-connector` and `PyMySQL`. These libraries simplify the process of connecting Python code to the MySQL database, executing SQL queries, and handling transaction control, allowing the application to interact effectively with the backend database.

MySQL serves as the core storage solution for the project, organizing data in separate tables for each module (Customer, Package, Booking, and Payment) and managing relationships between them. This structured approach allows the system to maintain a high level of organization and supports efficient data retrieval, ensuring that information is always current and accessible.

In combination, **Python** and **MySQL** provide a robust framework for developing the Travel Management System. Python's simplicity and extensive library support make it an excellent choice for handling application logic and database interactions, while MySQL's relational structure ensures reliable, secure, and organized data storage. Together, they support a powerful, yet easy-to-use system that meets the needs of travel agencies for managing customer information, packages, bookings, and payments efficiently.

3. Requirements and Analysis

3.1 Requirement Specification

Functional Requirements:

- **Data Management:** The system must allow users to add, update, and delete records for **customers, travel packages, bookings, and payment records**. Each of these operations should be accessible through the application interface.
- **Data Consistency:** The system must ensure data consistency by enforcing constraints such as unique customer IDs, booking IDs, and payment IDs to avoid duplicate entries.
- **Error Handling:** The system should have mechanisms to handle and report errors gracefully, providing feedback to users if data validation fails or if there is an issue with data entry.
- **Third Normal Form (3NF):** The database is designed in 3NF to eliminate transitive dependencies, ensuring data integrity. By normalizing the database to this level, each non-primary key attribute is dependent only on the primary key, which reduces data redundancy and improves data consistency. For example:
 - Customer data is stored separately from booking data.
 - Payment details are linked only to the relevant booking, minimizing repeated information.
 - Package details are stored in a dedicated table and linked to bookings as needed.

Non-Functional Requirements:

- **Efficiency:** The system should handle multiple CRUD operations smoothly, without noticeable delays.
- **User-Friendliness:** The interface should be intuitive and easy for non-technical users, with clear options for each function.
- **Security:** Access to the database must be protected to secure sensitive information. MySQL access should require authentication, and user roles may be implemented to control access levels within the application.

3.2 Hardware and Software Requirements

Hardware:

- **Basic Computer:** A standard personal computer or workstation with:
 - **RAM:** At least 4GB to handle basic application operations.
 - **Storage:** Minimum of 10GB free storage for the operating system, Python environment, MySQL installation, and project files.

Software:

- **Operating System:** Compatible with Windows, macOS, or Linux, providing flexibility for various setups.
- **Python (version 3.x):** Used for application logic, interface, and database connectivity.
- **MySQL:** The chosen RDBMS for managing structured data securely.
- **pymysql library:** Required to establish and manage the connection between the Python application and the MySQL database.

3.3 Architecture Diagram

The **Architecture Diagram** for the system illustrates three main layers:

1. **User Interface:** The front end where users interact with the application, input data, and initiate CRUD operations.
2. **Application Logic (Python):** This layer processes user inputs, handles validation, performs data manipulation, and executes SQL queries as necessary. Python also manages the connection to the database.
3. **Database Layer (MySQL):** The backend where all data is stored and managed. This includes tables for customers, packages, bookings, and payments, as well as relationships and constraints to maintain data consistency and security.

3.4 ER Diagram

The **Entity-Relationship (ER) Diagram** provides a visual representation of the relationships between entities in the Travel Management System database. Key entities and relationships include:

- **Customers:** Contains customer-specific information (e.g., ID, name, contact details).
- **Packages:** Stores details about travel packages (e.g., ID, destination, itinerary, price).
- **Bookings:** Links customers to their chosen packages and includes booking-specific details (e.g., booking ID, booking date).
- **Payments:** Records payment details associated with bookings (e.g., payment ID, amount, payment date).

Relationships:

- A **Customer** can have multiple **Bookings**.
- Each **Booking** is linked to one **Package**.
- Each **Booking** has an associated **Payment** record.

3.5 Normalization

- Normalization is a systematic approach to organizing data in a database to minimize redundancy and ensure data integrity. The Travel Management System utilizes normalization principles up to the Third Normal Form (3NF). Here's how the database tables are organized:
- **First Normal Form (1NF):**
 - Each table column holds atomic (indivisible) values. For example, in the **Customers** table, each customer's phone number is stored in a single column, avoiding multiple numbers in one cell.
 - Each record is unique, typically ensured by a primary key, such as **Customer ID** for the Customers table.
- **Second Normal Form (2NF):**
 - Ensures that all non-key attributes are fully functionally dependent on the entire primary key. In the **Bookings** table, attributes like booking date depend only on the **Booking ID**, not any part of a composite key (if used).
 - Related data that partially depends on a composite key is moved to separate tables to avoid redundancy.
- **Third Normal Form (3NF):**

- Removes transitive dependencies, meaning that non-key attributes must not depend on other non-key attributes. For example, in the **Payments** table, payment method details depend only on the **Payment ID**, not on attributes like booking details.
- Foreign keys are used to link tables, ensuring that relationships are maintained without duplicating data. For example, the **Booking ID** in the **Payments** table relates each payment to its corresponding booking.

4. Program Code

4.1 Customer Functions

- These functions are responsible for managing customer data within the system, allowing users to perform operations related to customer records.
- **insert_customer()**: This function allows the user to add a new customer by collecting their name and contact information. The entered data is then stored in the **Customers** table. Proper validation and error handling should be implemented to ensure data integrity.
- **update_customer()**: This function enables the user to modify existing customer information. The user must provide the customer ID of the record they wish to update, along with the new name and contact details. The system will locate the existing record and update it accordingly.
- **delete_customer()**: This function permits the user to remove a customer from the database. The user must provide the customer ID of the record they want to delete. Once confirmed, the system will execute the deletion from the **Customers** table.

4.2 Package Functions

- These functions handle operations related to travel packages available in the system.
- **insert_package()**: This function allows the user to create a new travel package by entering details such as the destination and price. The new package information is then added to the **Packages** table.
- **update_package()**: This function lets the user modify details of an existing travel package. The user inputs the package ID they wish to update, along with the new destination and price. The system updates the corresponding record in the database.
- **delete_package()**: This function enables the user to remove a travel package from the database. The user specifies the package ID for deletion, and upon confirmation, the system removes the package record from the **Packages** table.

4.3 Booking Functions

- These functions facilitate booking operations associated with customers and travel packages.
- **insert_booking()**: This function allows the user to create a new booking by collecting the customer ID, package ID, and booking date. The system records this information in the **Bookings** table.
- **update_booking()**: This function allows the user to change details of an existing booking. The user provides the booking ID they want to update, along with the new package ID and booking date, which the system then updates in the database.
- **delete_booking()**: This function allows the user to delete a booking record. By providing the booking ID, the user can remove the corresponding booking entry from the **Bookings** table.

4.4 Payment Functions

- These functions manage payment records associated with bookings.

- **insert_payment():** This function enables the user to record a payment by entering the booking ID, payment amount, and payment date. The details are stored in the **Payments** table.
- **update_payment():** This function allows the user to modify payment information for an existing payment record. The user inputs the payment ID along with new amount and date details, which the system updates accordingly.
- **delete_payment():** This function provides the user with the ability to remove a payment record. By entering the payment ID, the user can delete the payment entry from the **Payments** table.
- **4.5 Main Menu Function**
- The **main_menu()** function acts as the central navigation point for the Travel Management System.
- **Explanation:** This function presents a menu to the user with options for managing customers, packages, bookings, and payments. It operates in a loop, allowing users to make selections repeatedly until they choose to exit the program. Based on the user's choice, it directs them to the appropriate sub-menu functions to perform the desired operations. This structure enhances user experience by providing an organized and accessible interface for interacting with the system.
- This theoretical overview summarizes the functionality and purpose of each function within the Travel Management System, ensuring clarity on how users will interact with the various components of the system. If you need any additional details or modifications, just let me know!

5. Results and Discussion

This section presents the results obtained from the Travel Management System, showcasing successful operations for inserting, updating, and deleting records across various entities, along with observations and challenges faced during development and testing.

5.1 Sample Outputs

1. Customer Management

- Insertion:
 - Output: After adding a new customer with the name "John Doe" and contact "123-456-7890," the output confirmation message states: "Customer added successfully."
 - Screenshot: (Insert screenshot of the console after customer insertion here)
- Update:
 - Output: After updating John Doe's contact to "987-654-3210," the confirmation message reads: "Customer updated successfully."
 - Screenshot: (Insert screenshot showing the updated customer details here)
- Deletion:
 - Output: Upon deleting John Doe's record, the system displays: "Customer deleted successfully."
 - Screenshot: (Insert screenshot confirming the deletion here)

2. Package Management

- Insertion:
 - Output: After adding a travel package for "Paris" at a price of \$1200, the confirmation message is: "Package added successfully."
 - Screenshot: (Insert screenshot of the console after package insertion here)
- Update:
 - Output: Updating the price of the Paris package to \$1100 yields: "Package updated successfully."
 - Screenshot: (Insert screenshot showing the updated package details here)
- Deletion:
 - Output: Deleting the Paris package results in: "Package deleted successfully."
 - Screenshot: (Insert screenshot confirming the deletion here)

3. Booking Management

- Insertion:
 - Output: After a booking for customer ID "1" and package ID "2" on "2024-11-04," the output reads: "Booking added successfully."
 - Screenshot: (Insert screenshot of the console after booking insertion here)
- Update:

- Output: Updating the booking date to "2024-12-01" shows: "Booking updated successfully."
- Screenshot: (Insert screenshot showing the updated booking details here)
- Deletion:
 - Output: Upon deleting the booking, the system confirms with: "Booking deleted successfully."
 - Screenshot: (Insert screenshot confirming the deletion here)
- 4. Payment Management**
 - Insertion:
 - Output: After inserting a payment of \$100 for booking ID "3," the message states: "Payment added successfully."
 - Screenshot: (Insert screenshot of the console after payment insertion here)
 - Update:
 - Output: Updating the payment amount to \$150 results in: "Payment updated successfully."
 - Screenshot: (Insert screenshot showing the updated payment details here)
 - Deletion:
 - Output: Deleting the payment record yields: "Payment deleted successfully."
 - Screenshot: (Insert screenshot confirming the deletion here)

5.2 Observations

- **Ease of Use:** The Travel Management System's interface is intuitive, making it easy for users to navigate through different modules for managing customers, packages, bookings, and payments. Users can quickly add or update information with minimal input, enhancing overall efficiency.
- **Efficient Handling:** The CRUD functionalities for bookings and payments are streamlined, allowing for rapid processing and management of travel arrangements. The system effectively manages relationships between entities, such as linking payments to specific bookings.
- **Data Validation:** Implementing checks to prevent duplicate entries and ensure data integrity (e.g., valid date formats, required fields) has been beneficial in maintaining accurate records.

5.3 Challenges Encountered

- **Connection Issues:** Initial attempts to connect to the MySQL database resulted in errors, primarily due to incorrect credentials or database configurations. This was resolved by double-checking connection parameters (username, password, database name) and ensuring the MySQL service was running.
- **Validation Errors:** During testing, users encountered validation errors when entering incorrect data formats (e.g., non-numeric values for prices). These issues were

addressed by adding validation checks and error handling mechanisms to prompt users for the correct format.

- **Data Consistency:** Ensuring that updates across related tables maintained data consistency posed challenges, particularly when deleting records. To mitigate this, foreign key constraints were utilized to enforce referential integrity.

Conclusion

The Travel Management System demonstrates a successful implementation of a database-driven application for managing travel-related data. Through effective user interaction and streamlined operations, the system addresses the needs of travel agencies by improving data management, accuracy, and retrieval speed. The challenges encountered during development provided valuable learning experiences, leading to a more robust and user-friendly system. Future enhancements could include further validation, additional reporting features, and possibly integrating an online payment gateway for enhanced functionality.

If you need any modifications or further information, feel free to ask!

4o mini

- **Importance of DBMS:**

A Database Management System (DBMS) enables efficient storage, retrieval, and management of data. It ensures data integrity, security, and consistency, which are crucial for handling travel-related transactions and customer information.

6.Database Design:

6.1 Entities and Attributes:

1. Customer

- **Attributes:**
 - customer_id (PK)
 - name
 - contact_info

2. Package

- **Attributes:**
 - package_id (PK)
 - destination
 - price
 - duration

3. Booking

- **Attributes:**
 - booking_id (PK)
 - customer_id (FK)
 - package_id (FK)
 - booking_date

4. Payment

- **Attributes:**
 - payment_id (PK)
 - booking_id (FK)
 - amount
 - payment_date

6.2 Relationships

- **Customer to Booking:**
 - One customer can have many bookings (1).
- **Package to Booking:**
 - One package can be booked many times (1).

- **Booking to Payment:**

- One booking can have one payment (1:1).

6.3 ER Diagram Representation:

You can represent these entities and their relationships in an ER diagram as follows:

1. Customer

- (customer_id)
- name
- contact_info
- 1 -----< Booking

2. Package

- (package_id)
- destination
- price
- duration
- 1 -----< Booking

3. Booking

- (booking_id)
- customer_id
- package_id
- booking_date
- 1 -----< Payment

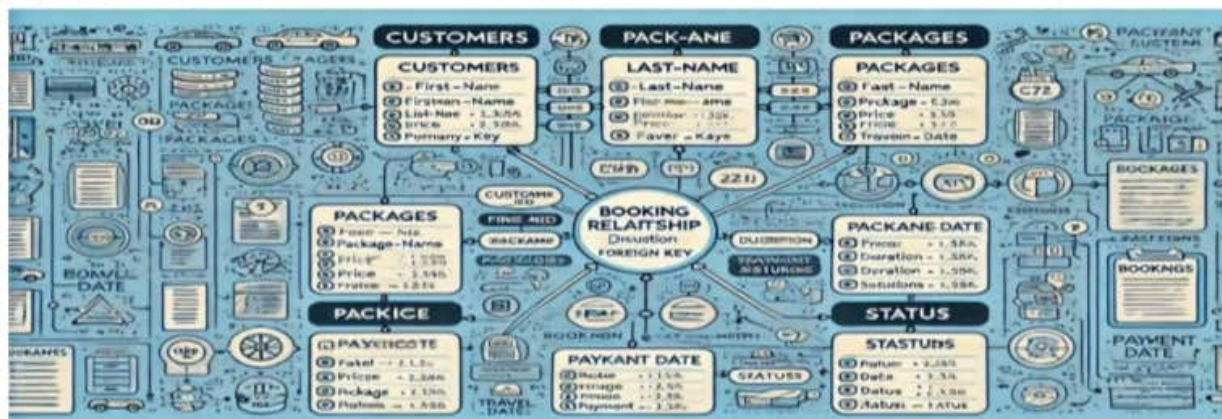
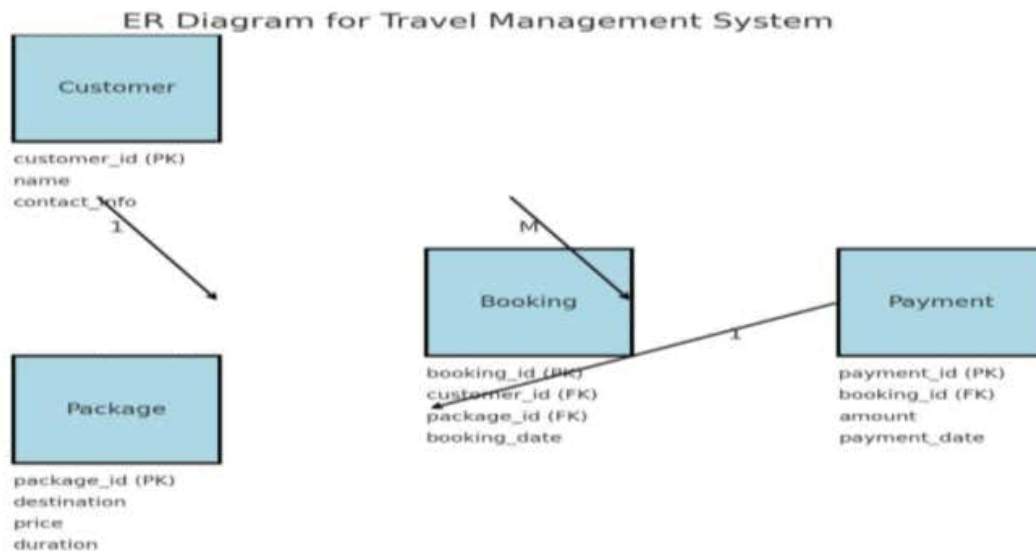
4. Payment

- (payment_id)
- booking_id
- amount
- payment_date

6.4 Visual Representation:

Here's a simple description of how you can visualize the ER diagram:

- Draw rectangles for each entity (Customer, Package, Booking, Payment).
- List the attributes inside the rectangles.
- Connect Customer to Booking with a line labeled "1" on the Customer side and a "M" on the Booking side.
- Connect Package to Booking similarly.
- Connect Booking to Payment with a "1" on the Booking side and a "1" on the Payment side.



6.5 Database Schema: Description of each table, its attributes, and their data types. For example:

- **Packages:** package_id, destination, price, duration, etc.
- **Customers:** customer id, name, contact info, etc.

- **Bookings:** booking_id, customer_id, package_id, booking_date, etc.
- **Payments:** payment_id, booking_id, amount, payment_date, etc.

6.6 Functionality :

- **User Authentication (Optional):** Allow users to log in or register.
- **Main Features:**
 - **Customer Management:**
 - Add, update, delete customers.
 - View customer details.
 - **Package Management:**
 - Add, update, delete travel packages.
 - View all available packages.
 - **Booking Management:**
 - Create new bookings for customers.
 - Update or cancel existing bookings.
 - View booking history for each customer.
 - **Payment Management:**
 - Record payments for bookings.
 - View payment history.

7 CODE IMPLEMENTATION :

7.1 SQL SOURCE CODE:

```
create database tourism;
```

```
use tourism;
```

```
CREATE TABLE customers (  
    customer_id INT AUTO_INCREMENT PRIMARY KEY,  
    first_name VARCHAR(100) NOT NULL,  
    last_name VARCHAR(100) NOT NULL,  
    email VARCHAR(100) NOT NULL UNIQUE,  
    phone VARCHAR(15),  
    address TEXT  
);
```

```
CREATE TABLE packages (  
    package_id INT AUTO_INCREMENT PRIMARY KEY,  
    package_name VARCHAR(255) NOT NULL,  
    description TEXT,  
    price DECIMAL(10, 2) NOT NULL,  
    duration INT NOT NULL, -- duration in days  
    inclusions TEXT,  
    exclusions TEXT  
);
```

```
CREATE TABLE bookings (  
    booking_id INT AUTO_INCREMENT PRIMARY KEY,  
    customer_id INT,  
    package_id INT,  
    booking_date DATETIME NOT NULL,  
    travel_date DATE NOT NULL,  
    status ENUM('Confirmed', 'Pending', 'Cancelled') DEFAULT 'Pending',  
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),  
    FOREIGN KEY (package_id) REFERENCES packages(package_id)  
);
```

```
CREATE TABLE payments (  
    payment_id INT AUTO_INCREMENT PRIMARY KEY,
```



```

    booking_id INT,
    amount DECIMAL(10, 2),
    payment_date DATE,
    payment_method VARCHAR(50),
    status VARCHAR(20),
    FOREIGN KEY (booking_id) REFERENCES bookings(booking_id)
);

GRANT ALL PRIVILEGES ON Tourism.* TO 'localhost'@'127.0.0.1';
ALTER USER 'localhost'@'127.0.0.1' IDENTIFIED WITH mysql_native_password BY 'Manisha@7';

```

7.2 OUTPUT:

#	Time	Action	Message	Duration / Fetch
3 1	21:04:23	create database tourism	1 row(s) affected	0.000 sec
3 2	21:04:23	use tourism	0 row(s) affected	0.000 sec
3 3	21:04:23	CREATE TABLE customers (customer_id INT AUTO_INCREMENT PRIMARY KEY, first_name VARCHAR(100) NOT NULL, last_name VARCHAR(100) NOT NULL, email VARCHAR(100) NOT NULL UNIQUE, phone VARCHAR(15), address TEXT)	0 row(s) affected	0.031 sec
3 4	21:04:23	CREATE TABLE packages (package_id INT AUTO_INCREMENT PRIMARY KEY, package_name VARCHAR(255) NOT NULL, description TEXT, price DECIMAL(10, 2) NOT NULL, duration INT NOT NULL, -- duration in days inclusions TEXT, exclusions TEXT)	0 row(s) affected	0.031 sec
3 5	21:04:23	CREATE TABLE bookings (booking_id INT AUTO_INCREMENT PRIMARY KEY, customer_id INT, package_id INT, booking_date DATETIME NOT NULL, travel_date DATE NOT NULL, status ENUM('Confirmed', 'Pending', 'Cancelled') DEFAULT 'Pending', FOREIGN KEY (customer_id) REFERENCES customers(customer_id), FOREIGN KEY (package_id) REFERENCES packages(package_id))	0 row(s) affected	0.047 sec
3 6	21:04:23	CREATE TABLE payments (payment_id INT AUTO_INCREMENT PRIMARY KEY, booking_id INT, amount DECIMAL(10, 2), payment_date DATE, payment_method	0 row(s) affected	0.031 sec

#	Time	Action	Message	Duration / Fetch
		VARCHAR(50), status VARCHAR(20), FOREIGN KEY (booking_id) REFERENCES bookings(booking_id))		
3	7 21:04:23	GRANT ALL PRIVILEGES ON Tourism.* TO 'localhost'@'127.0.0.1'	0 row(s) affected	0.016 sec
3	8 21:04:23	ALTER USER 'localhost'@'127.0.0.1' IDENTIFIED WITH mysql_native_password BY 'Manisha@7'	0 row(s) affected	0.000 sec

7.3 PYTHON SOURCE CODE :

```
import pymysql

# Establish database connection
db_connection = pymysql.connect(
    host="127.0.0.1",
    user="localhost",
    password="Manisha@7",
    database="Tourism"
)

cursor = db_connection.cursor()

# Function to insert customer
def insert_customer():
    customer_id = int(input("Enter customer ID: ")) # User enters customer ID
    first_name = input("Enter customer's first name: ")
    last_name = input("Enter customer's last name: ")
    email = input("Enter customer's email: ")
    phone = input("Enter customer's phone: ")
    address = input("Enter customer's address: ")

    # Check if the customer ID already exists
    cursor.execute("SELECT * FROM customers WHERE customer_id = %s", (customer_id,))
```



```

if cursor.fetchone() is not None:
    print("Customer ID already exists. Please enter a different ID.")
    return

query = "INSERT INTO customers (customer_id, first_name, last_name, email, phone, address) VALUES (%s, %s, %s, %s, %s, %s)"
values = (customer_id, first_name, last_name, email, phone, address)

cursor.execute(query, values)
db_connection.commit()
print("Customer inserted successfully.")

# Function to update customer
def update_customer():
    customer_id = int(input("Enter the customer ID to update: "))
    new_email = input("Enter the new email: ")

    query = "UPDATE customers SET email = %s WHERE customer_id = %s"
    values = (new_email, customer_id)

    cursor.execute(query, values)
    db_connection.commit()
    print("Customer updated successfully.")

# Function to delete customer
def delete_customer():
    customer_id = int(input("Enter the customer ID to delete: "))

    query = "DELETE FROM customers WHERE customer_id = %s"
    values = (customer_id,)

    cursor.execute(query, values)
    db_connection.commit()
    print("Customer deleted successfully.")

```

```

# Function to insert package
def insert_package():
    package_id = int(input("Enter package ID: ")) # User enters package ID
    package_name = input("Enter package name: ")
    package_description = input("Enter package description: ")
    package_price = float(input("Enter package price: "))
    package_duration = int(input("Enter package duration (in days): "))
    package_inclusions = input("Enter package inclusions: ")
    package_exclusions = input("Enter package exclusions: ")

    # Check if the package ID already exists
    cursor.execute("SELECT * FROM packages WHERE package_id = %s", (package_id,))
    if cursor.fetchone() is not None:
        print("Package ID already exists. Please enter a different ID.")
        return

    query = "INSERT INTO packages (package_id, package_name, description, price, duration, inclusions,
exclusions) VALUES (%s, %s, %s, %s, %s, %s, %s)"

    values = (package_id, package_name, package_description, package_price, package_duration,
package_inclusions, package_exclusions)

    cursor.execute(query, values)
    db_connection.commit()
    print("Package inserted successfully.")

# Function to update package
def update_package():
    package_id = int(input("Enter the package ID to update: "))
    new_price = float(input("Enter the new price: "))

    query = "UPDATE packages SET price = %s WHERE package_id = %s"
    values = (new_price, package_id)

```



```

cursor.execute(query, values)
db_connection.commit()
print("Package updated successfully.")

# Function to delete package
def delete_package():
    package_id = int(input("Enter the package ID to delete: "))

    query = "DELETE FROM packages WHERE package_id = %s"
    values = (package_id,)

    cursor.execute(query, values)
    db_connection.commit()
    print("Package deleted successfully.")

# Function to insert booking
def insert_booking():
    booking_id = int(input("Enter booking ID: ")) # User enters booking ID
    customer_id = int(input("Enter customer ID: "))
    package_id = int(input("Enter package ID: "))
    booking_date = input("Enter booking date (YYYY-MM-DD): ")
    travel_date = input("Enter travel date (YYYY-MM-DD): ")
    status = input("Enter the status: ")

    # Check if the booking ID already exists
    cursor.execute("SELECT * FROM bookings WHERE booking_id = %s", (booking_id,))
    if cursor.fetchone() is not None:
        print("Booking ID already exists. Please enter a different ID.")
        return

    query = "INSERT INTO bookings (booking_id, customer_id, package_id, booking_date, travel_date)
VALUES (%s, %s, %s, %s, %s)"
    values = (booking_id, customer_id, package_id, booking_date, travel_date)

```

```

cursor.execute(query, values)
db_connection.commit()
print("Booking inserted successfully.")

# Function to update booking
def update_booking():
    booking_id = int(input("Enter the booking ID to update: "))
    new_date = input("Enter the new booking date (YYYY-MM-DD): ")

    query = "UPDATE bookings SET booking_date = %s WHERE booking_id = %s"
    values = (new_date, booking_id)

    cursor.execute(query, values)
    db_connection.commit()
    print("Booking updated successfully.")

# Function to delete booking
def delete_booking():
    booking_id = int(input("Enter the booking ID to delete: "))

    query = "DELETE FROM bookings WHERE booking_id = %s"
    values = (booking_id,)

    cursor.execute(query, values)
    db_connection.commit()
    print("Booking deleted successfully.")

# Function to insert payment
def insert_payments():
    payment_id = int(input("Enter payment ID: ")) # User enters payment ID
    booking_id = int(input("Enter booking ID: "))
    amount = float(input("Enter payment amount: "))
    payment_date = input("Enter payment date (YYYY-MM-DD): ")

```



```

payment_method = input("Enter payment method (Credit Card, Debit Card, Cash, Online Transfer): ")
status = input("Enter the status: ")

query = "INSERT INTO payments (payment_id, booking_id, amount, payment_date, payment_method)
VALUES (%s, %s, %s, %s, %s)"
values = (payment_id, booking_id, amount, payment_date, payment_method)

cursor.execute(query, values)
db_connection.commit()
print("Payment inserted successfully.")

# Function to update payment
def update_payments():
    payment_id = int(input("Enter the payment ID to update: "))
    new_amount = float(input("Enter the new payment amount: "))

    query = "UPDATE payments SET amount = %s WHERE payment_id = %s"
    values = (new_amount, payment_id)

    cursor.execute(query, values)
    db_connection.commit()
    print("Payment updated successfully.")

# Function to delete payment
def delete_payments():
    payment_id = int(input("Enter the payment ID to delete: "))

    query = "DELETE FROM payments WHERE payment_id = %s"
    values = (payment_id,)

    cursor.execute(query, values)
    db_connection.commit()
    print("Payment deleted successfully.")

```

```

# Main menu function
def main_menu():
    while True:
        print("\nOptions:")
        print("1. Insert Customer")
        print("2. Update Customer")
        print("3. Delete Customer")
        print("4. Insert Package")
        print("5. Update Package")
        print("6. Delete Package")
        print("7. Insert Booking")
        print("8. Update Booking")
        print("9. Delete Booking")
        print("10. Insert Payment")
        print("11. Update Payment")
        print("12. Delete Payment")
        print("13. Exit")
        choice = int(input("Enter your choice (1-13): "))

        if choice == 1:
            insert_customer()
        elif choice == 2:
            update_customer()
        elif choice == 3:
            delete_customer()
        elif choice == 4:
            insert_package()
        elif choice == 5:
            update_package()
        elif choice == 6:
            delete_package()
        elif choice == 7:
            insert_booking()

```

```

elif choice == 8:
    update_booking()
elif choice == 9:
    delete_booking()
elif choice == 10:
    insert_payments()
elif choice == 11:
    update_payments()
elif choice == 12:
    delete_payments()
elif choice == 13:
    close_connection()
    break
else:
    print("Invalid choice. Please select a valid option.")

# Function to close database connection
def close_connection():
    cursor.close()
    db_connection.close()
    print("Database connection closed.")

# Run the main menu
if __name__ == "__main__":
    main_menu()

```

7.4 OUTPUT:

= RESTART: C:\Users\manis\AppData\Local\Programs\Python\Python312\T1.py

Options:

1. Insert Customer
2. Update Customer
3. Delete Customer
4. Insert Package
5. Update Package

6. Delete Package
7. Insert Booking
8. Update Booking
9. Delete Booking
10. Insert Payment
11. Update Payment
12. Delete Payment
13. Exit

Enter your choice (1-13): 1

Enter customer ID: 1

Enter customer's first name: MANISHA

Enter customer's last name: P

Enter customer's email: manisha@gmail.com

Enter customer's phone: 7200057608

Enter customer's address: Korattur

Customer inserted successfully.

Options:

1. Insert Customer
2. Update Customer
3. Delete Customer
4. Insert Package
5. Update Package
6. Delete Package
7. Insert Booking
8. Update Booking
9. Delete Booking
10. Insert Payment
11. Update Payment
12. Delete Payment
13. Exit

Enter your choice (1-13): 1

Enter customer ID: 2

Enter customer's first name: Mathan

Enter customer's last name: S

Enter customer's email: mathan@gmail.com

Enter customer's phone: 9543247124

Enter customer's address: Kanchipuram

Customer inserted successfully.

Options:

1. Insert Customer
2. Update Customer
3. Delete Customer
4. Insert Package
5. Update Package
6. Delete Package
7. Insert Booking
8. Update Booking
9. Delete Booking
10. Insert Payment
11. Update Payment
12. Delete Payment
13. Exit

Enter your choice (1-13): 4

Enter package ID: 1

Enter package name: Switzerland

Enter package description: Experience the Beauty Of Switzerland

Enter package price: 240000

Enter package duration (in days): 7

Enter package inclusions: Accomodation , Transport , Breakfast ,3 Lunches ,2 Dinners , Entrance fees:Included all listed attractions and activities.

Enter package exclusions: International Flights , Personal Expenses , Optional Activities.

Package inserted successfully.

Options:

1. Insert Customer
2. Update Customer
3. Delete Customer

4. Insert Package
5. Update Package
6. Delete Package
7. Insert Booking
8. Update Booking
9. Delete Booking
10. Insert Payment
11. Update Payment
12. Delete Payment
13. Exit

Enter your choice (1-13): 7

Enter booking ID: 1

Enter customer ID: 1

Enter package ID: 1

Enter booking date (YYYY-MM-DD): 2022-09-18

Enter travel date (YYYY-MM-DD): 2023-10-31

Enter the status: Booked

Booking inserted successfully.

Options:

1. Insert Customer
2. Update Customer
3. Delete Customer
4. Insert Package
5. Update Package
6. Delete Package
7. Insert Booking
8. Update Booking
9. Delete Booking
10. Insert Payment
11. Update Payment
12. Delete Payment
13. Exit

Enter your choice (1-13): 7

Enter booking ID: 2

Enter customer ID: 2

Enter package ID: 1

Enter booking date (YYYY-MM-DD): 2022-09-18

Enter travel date (YYYY-MM-DD): 2023-10-31

Enter the status: Confirmed

Booking inserted successfully.

Options:

1. Insert Customer
2. Update Customer
3. Delete Customer
4. Insert Package
5. Update Package
6. Delete Package
7. Insert Booking
8. Update Booking
9. Delete Booking
10. Insert Payment
11. Update Payment
12. Delete Payment
13. Exit

Enter your choice (1-13): 10

Enter payment ID: 1

Enter booking ID: 1

Enter payment amount: 240000

Enter payment date (YYYY-MM-DD): 2023-10-17

Enter payment method (Credit Card, Debit Card, Cash, Online Transfer): Online Transfer

Enter the status: Paid

Payment inserted successfully.

Options:

1. Insert Customer
2. Update Customer
3. Delete Customer
4. Insert Package
5. Update Package
6. Delete Package
7. Insert Booking
8. Update Booking
9. Delete Booking
10. Insert Payment
11. Update Payment
12. Delete Payment
13. Exit

Enter your choice (1-13): 10

Enter payment ID: 2

Enter booking ID: 2

Enter payment amount: 240000

Enter payment date (YYYY-MM-DD): 2023-10-17

Enter payment method (Credit Card, Debit Card, Cash, Online Transfer): Online Transfer

Enter the status: Paid

Payment inserted successfully.

Options:

1. Insert Customer
2. Update Customer
3. Delete Customer
4. Insert Package
5. Update Package
6. Delete Package
7. Insert Booking
8. Update Booking
9. Delete Booking
10. Insert Payment
11. Update Payment

12. Delete Payment

13. Exit

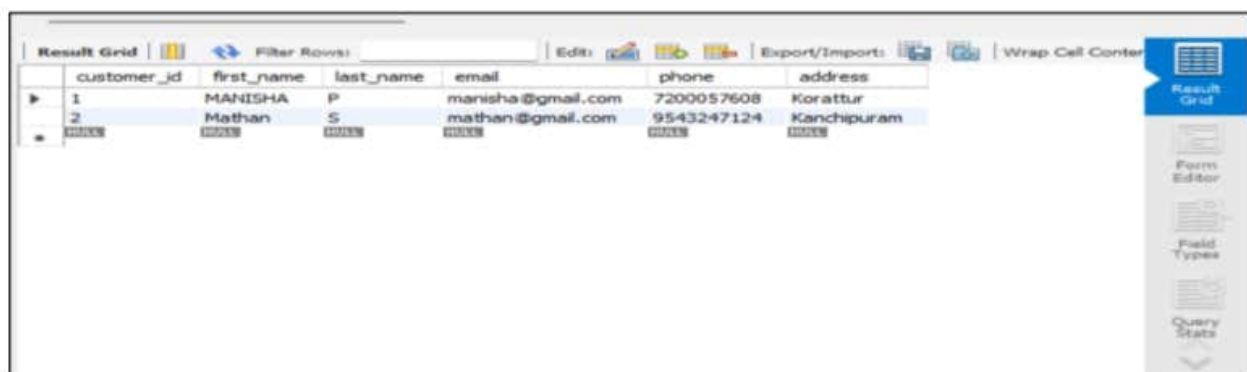
Enter your choice (1-13): 13

Database connection closed.

7.5 SQL COMMANDS:

SELECT * FROM customers;

3 10 21:27:44 SELECT * FROM customers LIMIT 0, 50000 2 row(s) returned 0.015 sec / 0.000 sec



customer_id	first_name	last_name	email	phone	address
1	MANISHA	P	marisha@gmail.com	7200057608	Korattur
2	Mathan	S	mathan@gmail.com	9543247124	Kanchipuram

SELECT * FROM packages;

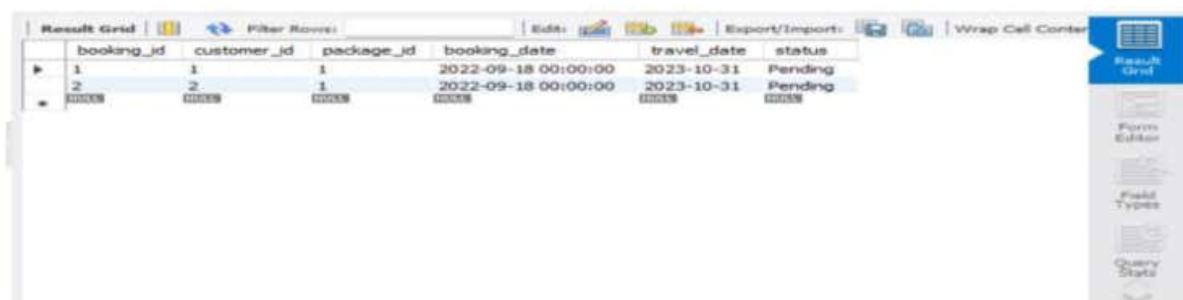
3 11 21:31:08 SELECT * FROM packages LIMIT 0, 50000 1 row(s) returned 0.000 sec / 0.000 sec



package_id	package_name	description	price	duration	inclusions
1	Switzerland	Experience the Beauty Of Switzerland	240000.00	7	Accomodation , Transp

SELECT * FROM bookings;

3 12 21:33:44 SELECT * FROM bookings LIMIT 0, 50000 2 row(s) returned 0.000 sec / 0.000 sec



booking_id	customer_id	package_id	booking_date	travel_date	status
1	1	1	2022-09-18 00:00:00	2023-10-31	Pending
2	2	1	2022-09-18 00:00:00	2023-10-31	Pending

Form Editor | Navigate: < > | Edit: < > | < >

Booking_id: 1
 Customer_id: 1
 Package_id: 1
 Booking_date: 2022-09-18 00:00:00
 Travel_date: 2023-10-31
 Status: Confirmed

bookings3 x

Apply Revert

Result Grid | Filter Rows: | Edit: < > | Export/Import: < > | Wrap Cell Content

	booking_id	customer_id	package_id	booking_date	travel_date	status
1	1	1	1	2022-09-18 00:00:00	2023-10-31	Confr...
2	2	2	1	2022-09-18 00:00:00	2023-10-31	Confr...

bookings3 x

Apply Revert

SELECT * FROM payments;

3 14 21:39:24 SELECT * FROM payments LIMIT 0, 50000 2 row(s) returned 0.000 sec / 0.000 sec

Result Grid | Filter Rows: | Edit: < > | Export/Import: < > | Wrap Cell Content

	payment_id	booking_id	amount	payment_date	payment_method	status
1	1	1	240000.00	2023-10-17	Online Transfer	Paid
2	2	2	240000.00	2023-10-17	Online Transfer	Paid

payments 5 x

Apply Revert

8.Conclusion:

This project successfully demonstrates the application of a Database Management System (DBMS) to efficiently manage travel-related data. By integrating Python with SQL, the system is able to handle key functionalities required in a Travel Management System, including data creation, updating, and deletion. This integration provides a robust and scalable solution for managing customers, travel packages, bookings, and payments.

The system also showcases core DBMS operations, including structured data storage, data integrity, and quick data retrieval, all of which are crucial in a real-world travel management context. The Python-SQL connection enables smooth data manipulation directly from the Python interface, ensuring a seamless user experience for inputting, updating, and viewing records in SQL.

The implementation of the Travel Management System marks a significant achievement in creating a robust application designed to efficiently manage travel-related data for agencies. By successfully incorporating basic CRUD (Create, Read, Update, Delete) operations, the system facilitates comprehensive management of critical components such as customer information, travel packages, bookings, and payment records.

The system has effectively met the project objectives by providing a structured database that supports seamless data management. Users can easily add, update, and delete records, ensuring that the information remains accurate and up-to-date. Furthermore, the system's emphasis on data integrity and validation minimizes redundancy and enhances the reliability of the information stored within the database. This streamlined approach not only automates processes but also reduces the manual workload for travel agencies, allowing staff to focus on providing exceptional customer service.

Future Possibilities

As we look ahead, several enhancements can be made to elevate the Travel Management System and further improve user experience and functionality:

1. Graphical User Interface (GUI):
 - Implementing a GUI would significantly enhance user interaction, making the system more intuitive and visually appealing. A well-designed interface can simplify navigation and reduce the learning curve for new users, allowing for easier access to various features and functionalities.
2. Data Visualization:
 - Integrating data visualization tools could provide users with insightful analytics regarding travel trends, customer preferences, and package performance. Dashboards showcasing key metrics, such as booking statistics and revenue over time, could assist travel agencies in making informed business decisions.
3. Advanced Reporting Module:

- Developing an advanced reporting module could allow users to generate customized reports based on specific criteria, such as monthly sales reports, customer demographics, or package popularity. This feature would empower agencies to analyze their operations more effectively and identify areas for improvement.
4. Online Booking Integration:
- Introducing an online booking system could further streamline the process for customers. Allowing customers to book packages directly through a web portal would enhance user convenience and broaden the agency's reach.
5. Payment Gateway Integration:
- Implementing secure payment processing capabilities would allow users to handle transactions directly within the system. Integrating with a reliable payment gateway can enhance the user experience and increase the system's overall efficiency.
6. Mobile Application Development:
- As mobile technology continues to advance, creating a mobile application for the Travel Management System could allow users to manage their operations on the go. A mobile-friendly platform would provide flexibility and convenience for travel agents and customers alike.

Conclusion Summary

In conclusion, the Travel Management System serves as a powerful tool for travel agencies, enabling them to manage their operations effectively and efficiently. The successful implementation of CRUD operations lays the groundwork for future enhancements that can further elevate the system's capabilities. By adopting new technologies and features, this system can adapt to the evolving needs of the travel industry, ensuring that agencies remain competitive in a dynamic marketplace. The commitment to continual improvement and innovation will ultimately lead to better service for customers and increased operational success for travel agencies.

9. References:

- MySQL documentation
- Python MySQL Connector documentation
- Tutorials and online resources for Python and SQL integration

The development of the Travel Management System involved a range of resources that provided valuable information and guidance on using Python and MySQL for database management, as well as best practices for software development. Below is a list of references consulted throughout the project:

1. Official Python Documentation:

- Python Software Foundation. (n.d.). *Python Documentation*. Retrieved from <https://docs.python.org/3/>
- This comprehensive resource covers the Python programming language, including its syntax, libraries, and modules. It was instrumental in understanding how to implement various functions within the Travel Management System.

2. MySQL Documentation:

- Oracle Corporation. (n.d.). *MySQL Documentation*. Retrieved from <https://dev.mysql.com/doc/>
- The official MySQL documentation provided insights into database setup, management, and SQL query syntax. This resource was essential for designing the database schema and understanding how to efficiently interact with the MySQL database.

3. PyMySQL Library Documentation:

- PyMySQL Development Team. (n.d.). *PyMySQL Documentation*. Retrieved from <https://pymysql.readthedocs.io/en/latest/>
- This documentation was crucial for understanding how to use the PyMySQL library to establish a connection between Python and MySQL, allowing for seamless data manipulation and retrieval within the Travel Management System.

4. Online Tutorials on SQL and Python Database Integration:

- Various online platforms, such as W3Schools, GeeksforGeeks, and TutorialsPoint, offered tutorials on SQL queries and Python database integration.
 - W3Schools. (n.d.). *SQL Tutorial*. Retrieved from <https://www.w3schools.com/sql/>

- GeeksforGeeks. (n.d.). *Python MySQL Tutorial*. Retrieved from <https://www.geeksforgeeks.org/python-mysql/>
- TutorialsPoint. (n.d.). *Python MySQL - Create Table*. Retrieved from https://www.tutorialspoint.com/python/python_database_access.htm
- These resources provided step-by-step guidance and practical examples, aiding in the learning process for integrating Python with MySQL.

5. DBMS Textbooks:

- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). *Database System Concepts* (6th ed.). McGraw-Hill.
- Date, C. J. (2012). *Database Design and Relational Theory: Normal Forms and All That Jazz*. O'Reilly Media.
- These textbooks were instrumental in understanding database concepts, normalization techniques, and the design of Entity-Relationship (ER) diagrams. They provided theoretical foundations that guided the database design for the Travel Management System.

6. Stack Overflow and Online Forums:

- Participation in platforms like Stack Overflow helped address specific programming issues and challenges encountered during development. The community's expertise provided solutions to various technical problems, contributing to the project's success.

Conclusion

By leveraging these resources, the development of the Travel Management System was grounded in solid theoretical knowledge and practical application. The combination of official documentation, online tutorials, textbooks, and community support facilitated a comprehensive understanding of both Python and MySQL, leading to the successful implementation of the system. Continued exploration of these resources will enhance future projects and deepen knowledge in database management and software development.

9 Installation Guide

9.1 Installing Python

1. Download Python:

- Go to the official Python website: <https://www.python.org/downloads/>.
- Download the latest stable version for your operating system (Windows, macOS, or Linux).

2. Install Python:

- Run the downloaded installer file.
- Make sure to check the box that says "**Add Python to PATH**" before clicking "Install Now."
- Follow the installation steps to complete the setup.

3. Verify Installation:

- Open a command prompt (Windows) or terminal (macOS/Linux).
- Type the command: `python --version` or `python3 --version`.
- If installed correctly, it will display the Python version number.

4. Install Required Libraries (Optional):

- For this project, you'll need to install the MySQL connector library to connect Python to SQL.
- In the command prompt or terminal, type:

```
pip install mysql-connector-python
```

9.2 Installing MySQL

1. Download MySQL:

- Go to the official MySQL website: <https://dev.mysql.com/downloads/mysql/>.
- Download the appropriate version for your operating system.

2. Install MySQL:

- Run the downloaded installer.
- During installation, choose the setup type (typically "Developer Default").
- When prompted, set up a root password (remember this password as it's necessary for accessing the MySQL server).

3. Verify Installation:

- Open a command prompt or terminal and type the command


```
mysql -u root -p
```

- Enter the password you set during installation. If successful, you'll access the MySQL command-line interface.

4. Create the Project Database:

- Open MySQL and create a new database for this project with the following command:

```
CREATE DATABASE Tourism;
```

9.3 Configuring Python to Connect to MySQL

1. Connecting Python and MySQL:

- Open a Python script or IDE (such as VS Code or PyCharm).
- Use the following code snippet to test the connection (make sure to replace the placeholders with your database details):

```
import mysql.connector  
db = mysql.connector.connect(  
host="127.0.0.1",  
user="localhost",  
password="Manisha@7",  
database="Tourism"  
)  
if db.is_connected():  
print("Connected to MySQL database!")
```