**[TOURISM MANAGEMENT SYSTEM]**

**Database Design Document**

**V 3.0**

**By**

|  |  |  |
| --- | --- | --- |
| **1** | **SHEHZANA BIBI** | **44** |
| **2** | **JALAL UR REHMAN** | **42** |
| **3** | **MUHAMMAD MUNAWAR KHAN** | **13** |



**Department of Computer Sciences**

**Namal University**

**Mianwali, Pakistan**

**Submission Date : 30th June, 2024**

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Approved by** |
| 10/6/24 | V.3.0 | Changes can be done in ERD as well as according to Data-Dictionary and entities relationships. | **Maam Asiya** |
| 22/04/24 | V 2.0 | No any change in previous document | **Maam Asiya** |

**TABLE OF CONTENTS**

[Chapter 1: Project Overview 4](#_Toc167466652)

[1.1. Introduction: 4](#_Toc167466653)

[1.2. Problem Statement: 4](#_Toc167466654)

[1.3. Project Objectives: 4](#_Toc167466655)

[1.4. Document Objectives: 4](#_Toc167466656)

[Chapter 2: Detailed Database Design 5](#_Toc167466657)

[2.1. Entity: 5](#_Toc167466659)

[2.2. Data Dictionary: 5](#_Toc167466660)

[2.3. Relationships: 5](#_Toc167466661)

[2.4. Entity Relationship Diagram: 6](#_Toc167466662)

[Chapter 3: Logical Database Design 7](#_Toc167466663)

[3.1. Relational Schema: 7](#_Toc167466664)

[3.2. Functional Dependencies: 7](#_Toc167466665)

[3.3. Normalization: 7](#_Toc167466666)

[Chapter 4: Physical Database Design 8](#_Toc167466667)

[4.1. Structure of the Tables: 8](#_Toc167466668)

[4.2. Data Samples Inside Tables: 8](#_Toc167466669)

[4.3. Queries Results: 8](#_Toc167466670)

[References 9](#_Toc167466671)

# PROJECT OVERVIEW

## INTRODUCTION:

In this project, we are going to make the website of the TOURISM MANAGEMENT SYSTEM which organizes the information on the booking, hotels, destinations, and the activities that will place. This website will provide a platform for tourists to visit the beautiful places of Pakistan. The database will store and manage the data efficiently like who is booking what, where they are traveling, and also where they will stay.

## PROBLEM STATEMENT:

The problems that have arisen and been tackled by the system is;

Current travel booking methods are characterized by inefficiency and frustration. Limited access to comprehensive information, time-consuming agency visits, and inconsistent service levels act as significant deterrents for travelers, hindering effective trip planning. This fragmented approach stifles the exploration of new destinations and ultimately impacts the growth of the tourism industry. A modern solution is imperative. A user-friendly online platform that offers comprehensive information, real-time availability, and personalized recommendations is required to streamline travel booking and enhance the overall planning experience."

**1.3. PROJECT OBJECTIVES:**

Objectives of the project are here;

* Make booking easy for travelers
* Give personalized suggestions with tour packages
* Keep payment and data safe
* Provide one place for booking and managing trips
* Provide online transaction, transportation, and accommodation

## 1.4. DOCUMENT OBJECTIVES:

The objective of the tourism management system documentation is to explain what the project is about, how its database is set up, what it can do[1] (like booking trips and planning itineraries), and how users interact with it. It should also cover any rules or guidelines for using the system and provide helpful instructions for building and maintaining it. Also it provides information about the website which will be helpful to every person.

# DETAILED DATABASE DESIGN



## ENTITY:

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Entity Name** | **Description** |
| 01 | Tourist | Takes the personal information of the tourists |
| 02 | TourPackages | Takes the information of the package that is selected by the tourist. |
| 03 | Destination | Provides information about the place where the tourist wants to visit. |
| 04 | Accommodation | Stores information about the hotels booked by the tourists. |
| 05 | PaymentTransaction | Records payment-related information made by the tourists. |
| 06 | Transportation | Stores information about the mode of transportation selected by the tourists. |
| 07 | Journey | Associative entity that records the connection between the transportation and the tourist. |
| 08 | Route | Associative entity that links destination and transportation. |
| 09 | Booking | Associative entity that records the Booking between the tourist, tour package and payment Transaction. |
| 10 | Stay | Associative entity that details the stay of the tourist in accommodations. |
| 11 | VisitRecord | Associative entity that links between the tourist and the destination. |
| 12 | PackagePrice | Tell us about the price according to type of package selected |

## DATA DICTIONARY:

* + 1. **Tourist:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | T\_ID | INT | PRIMARY KEY | Set of all possible tourist ID |
| 02 | Name | VARCHAR(255) | NOT NULL | Name of the tourist |
| 03 | Gender | CHAR(1) | NOT NULL | Gender of the tourist |
| 04 | Age | INT | NOT NULL | Age of tourist |
| 05 | Phone\_No | VARCHAR(255) | NULL | Phone number of the tourist |
| 06 | Email | VARCHAR(255) | NULL | Email address of the tourist |
| 07 | CNIC | VARCHAR(20) | NULL | CNIC of tourist |

### **2.2.2**. **Destination:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | Dest\_ID | INT | PRIMARY KEY | Destination ID |
| 02 | Name | VARCHAR(255) | NOT NULL | Name of the destination that tourists want to visit |
| 03 | Depar\_Location | VARCHAR(255) | NULL | Starting Location |
| 04 | Dest\_Location | VARCHAR(255) | NULL | Ending Location |

## 2.2.3. Accomodation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| *01* | Acc\_ID | INT | PRIMARY KEY | Accommodation ID |
| *02* | Dest\_ID | INT | FOREIGN | References Destination(Dest\_ID) |
| *03* | Hotel\_ Name | VARCHAR(255) | NOT NULL | Name of the hotel where tourists want to stay |

**2.2.4. Transportation:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | Trans\_ID | INT | PRIMARY KEY | Tourist transportation ID |
| 02 | Type | VARCHAR(255) | NULL | Type of transportation |
| 03 | Depar\_Time | TIME | NULL | Departure time |
| 04 | Arriv\_Time | TIME | NULL | Arrival time |

**2.2.5. PaymentTransaction:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | Pay\_ID | INT | PRIMARY KEY | Tourist payment transactionID |
| 02 | Payment\_Date | Date | NOT NULL | Date of the payment sent by the tourist |
| 03 | Payment\_Method | VARCHAR(255) | NOT NULL | Type of the transaction method(easy paisa,jazz cash, or any account like HBL) |
| 04 | Payment\_Status | VARCHAR(255) | NULL | Status like completed or pending. |

**2.2.6. PackagePrice:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01 | Type | VARCHAR(255) | PRIMARY KEY | Type of package |
| 02 | Price | DECIMAL(10,2) | NOT NULL | Price of the package |

**2.2.7. TourPackages:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01 | TP\_ID | INT | PRIMARY KEY | Tour Package ID |
| 02 | Package\_Name | VARCHAR(255) | NOT NULL | Name of the package selected by the tourist |
| 03 | Type | VARCHAR(255) | FOREIGN KEY | References PackagePrice(Type) |

**2.2.8. Route:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | Dest\_ID | INT | FOREIGN KEY | References destination(Dest\_ID) |
| 02 | Trans\_ID | INT | FOREIGN KEY | References transportation(Trans\_ID) |

**2.2.9. Journey:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | T\_ID | INT | FOREIGN KEY | References Tourist(T\_ID) |
| 02 | Trans\_ID | INT | FOREIGN KEY | References Transportation(Trans\_ID) |
| 03 | Depar\_Date | DATE | NULL | Departure date |

**2.2.10. Booking:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | B\_ID | INT | PRIMARY KEY AUTO INCREMENT | Composite key of the Booking of TP\_ID and T\_ID |
| 02 | T\_ID | INT | FOREIGN KEY | References Tourist(T\_ID) |
| 03 | TP\_ID | INT | FOREIGN KEY | References TourPackages(TP\_ID) |
| 04 | Pay\_ID | INT | FOREIGN KEY | References Payment\_Transaction(Pay\_ID) |
| 05 | Duration\_Ofpackage | INT | NULL | Duration of package in days |

**2.2.11. Stay:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | T\_ID | INT | FOREIGN KEY | References Tourist(T\_ID) |
| 02 | Acc\_ID | INT | FOREIGN KEY | References Accomodation(Acc\_ID) |
| 03 | Check\_In | DATE | NOT NULL | Check in date |
| 04 | Check\_Out | DATE | NOT NULL | Check out date |
| 05 | Room\_Type | VARCHAR(255) | NULL | Type of the room |

**2.2.12. VisitRecord:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | Dest\_ID | INT | FOREIGN KEY | References Destination(Dest\_ID) |
| 02 | T\_ID | INT | FOREIGN KEY | References Tourist(T\_ID) |
| 03 | Visit\_Date | DATE | NULL | Date of visit |

## RELATIONSHIPS:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Participating Entities** | **Relation** | **Business Rule** |
| 01 | Destination, tourist | Tourist visit destination | Many tourists visit many destinations and many destinations are visited by many tourists |
| 02 | Tourist, Accommodation | Tourist select accommodation | Many tourists select many accommodations and many accommodations can be selected by many tourists |
| 03 | Tourist, Tour\_packages, Payment\_Transaction | Tourist makes payment transaction for reserved tour packages | Many tourists reserve many payment transaction for many reserved tour packages and many tour packages reserve payment transaction for many tourists |
| 04 | Tourist, transportation | Tourist use transportaion | Many tourists use many transportations and many transportations used by many tourists |
| 05 | Destination, accommodation | The destination is associated with accommodation | One destination is associated with many accommodations and many accommodations are associated to only one destination |
| 06 | Destination, transportation | Destination is served by transportation | Many destinations are served by many transportations and many transportations serves for many destinations |

## ENTITY RELATIONSHIP DIAGRAM:

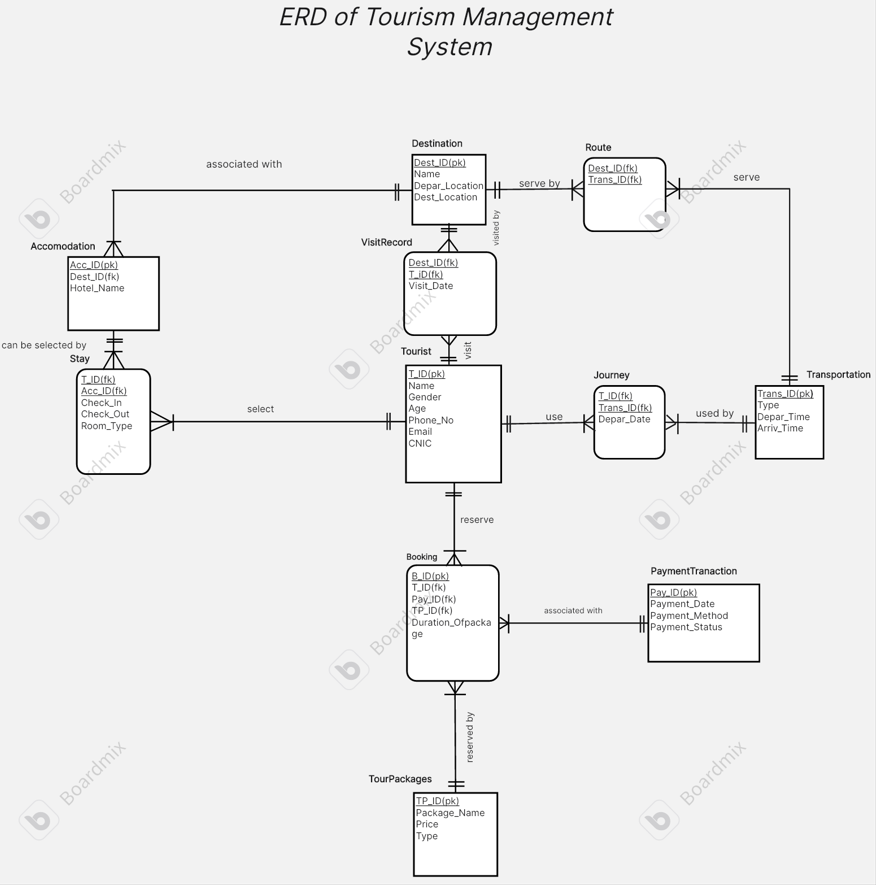
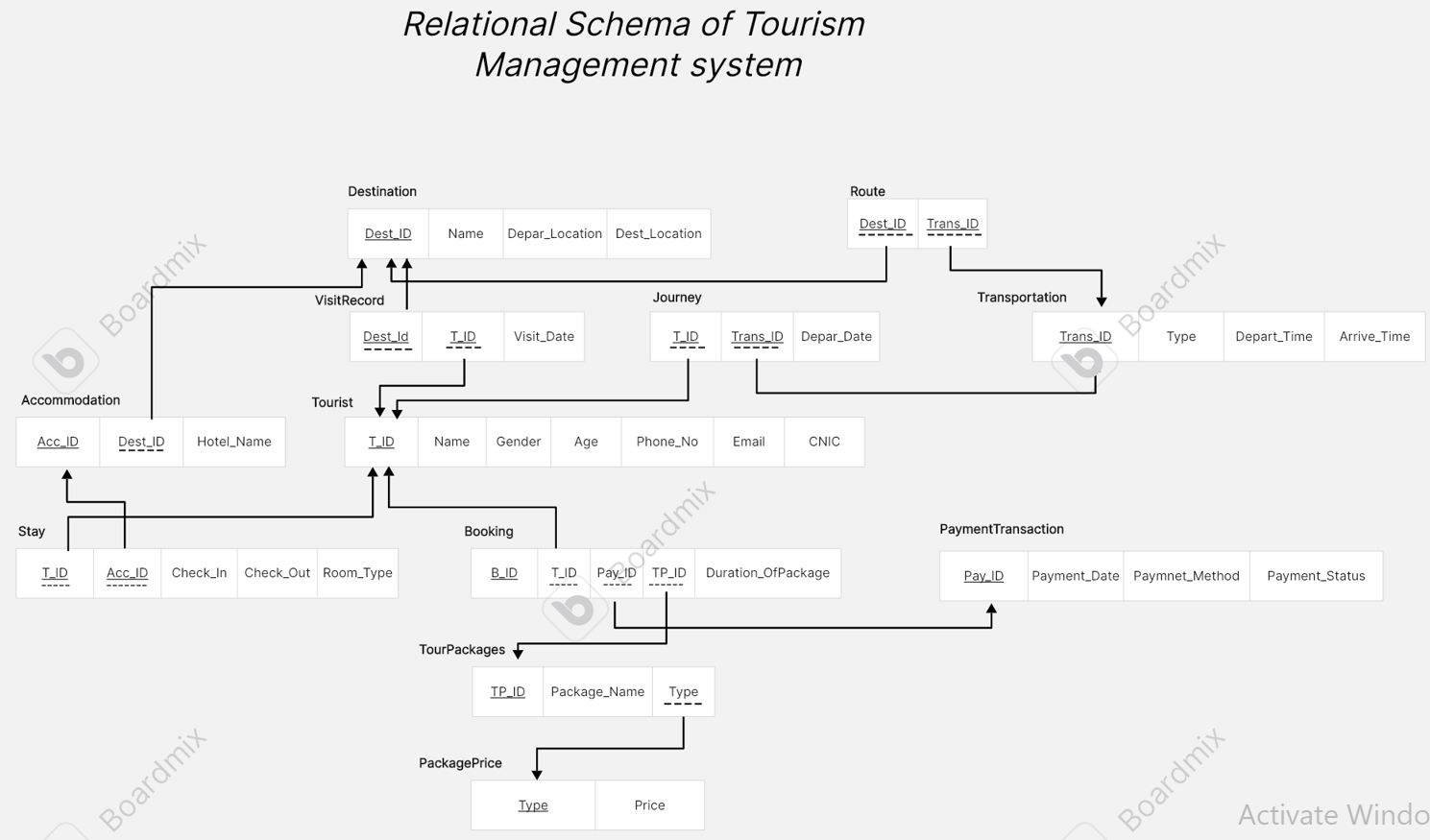


Figure 1: Tourism Management System [2]

# : Logical DATABASE DESIGN

* 1. **RELATIONAL SCHEMA:**

****

* 1. **FUNCTIONAL DEPENDENCIES:**

1. **Destination**

Dest\_ID 🡪 Name, Dest\_Location, Depart\_Locaion

1. **VisitRecord**

Dest\_ID,T\_ID 🡪 Visit\_Date

1. **Journey**

T\_ID,Trans\_ID 🡪 Depar\_Date

1. **Transportation**

Trans\_ID 🡪 Type, Depar\_Time, Arrive\_Time

1. **Accommodation**

Acc\_ID 🡪 Hotel\_Name

1. **Tourist**

T\_ID 🡪 Name, Gender, Age, CNIC, Email, Phone\_No

1. **Stay**

T\_ID,Acc\_ID 🡪 Check\_In, Check\_Out, Room\_Type

1. **Booking**

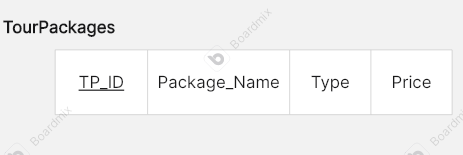
B\_ID 🡪 Duration\_OfPackage

1. **payment transaction**

Pay\_ID 🡪 Payment\_Date, Payment\_Method, Payment\_Status

1. **TourPackage**  
   TP\_ID 🡪 Pacage\_Name
2. **PackagePrice**

Type 🡪 Price



Only one transitive dependency exists which is price depends on type and type depends on TP\_ID.

TP\_ID->TYPE->Price

**3.3 NORMALIZATION:**

**1NF:**

No any step to convert my relational schema to 1NF because it is already present in 1NF form.

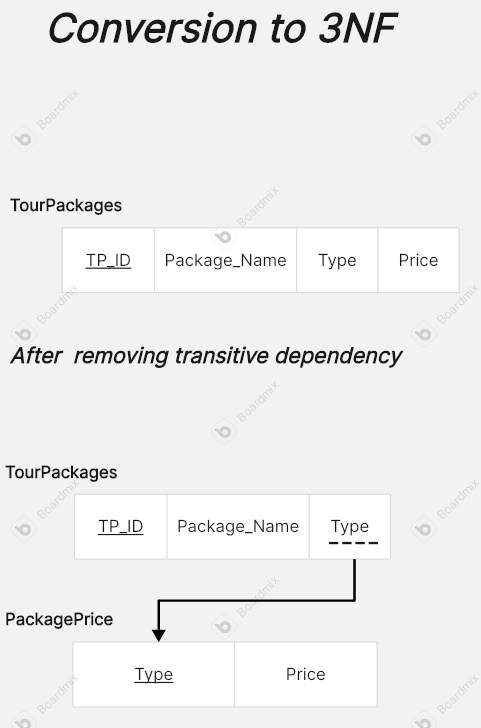
**2NF:**

There is no need to convert relational schema to 2NF because no any partial dependency exist in it.

**3NF:**

Convertion of transitive entity to 3NF.

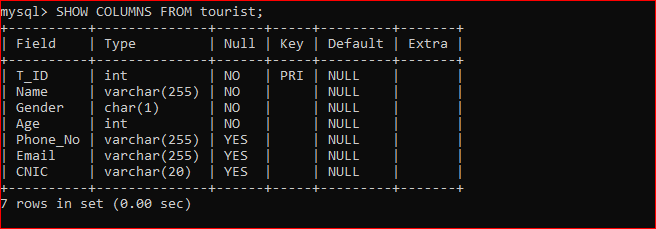
There is only one transitive entity exist which is price depends on type and type depends on tourist package\_id which can be removed given below:

**

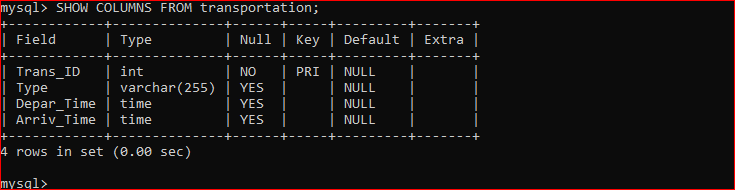
# : Physical DATABASE DESIGN

* 1. **STRUCTURE OF THE TABLE:**

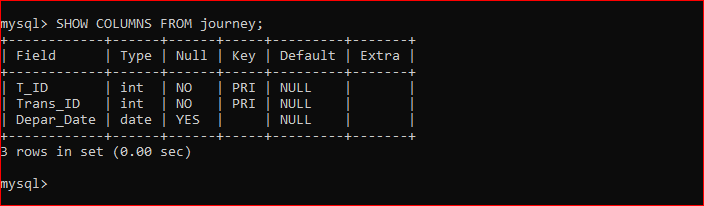
**1. Tourist**



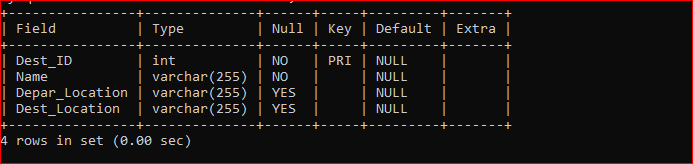
**2. Transportation:**



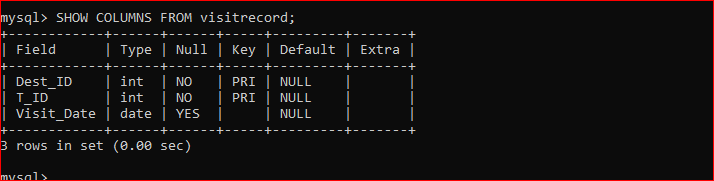
**3. Journey:**



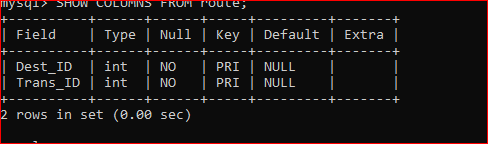
**4. Destination:**



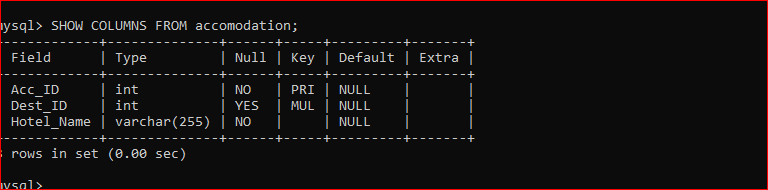
**5. Visitrecord:**



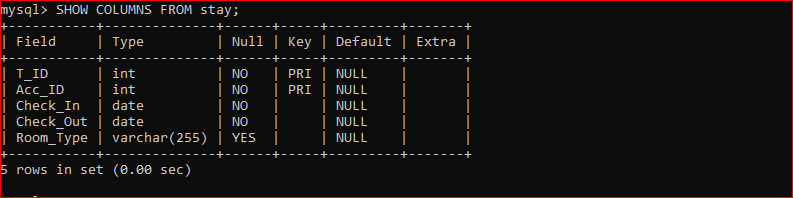
**6. Route:**



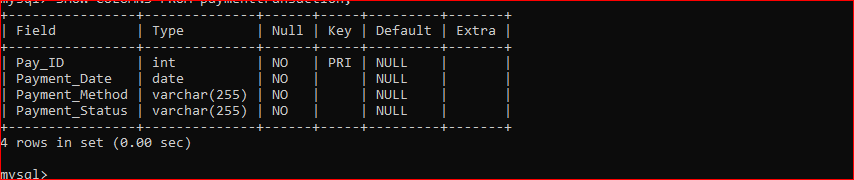
**7. Accomodation:**



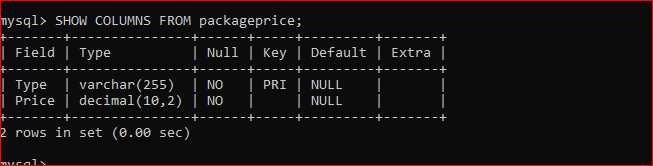
**8. Stay:**



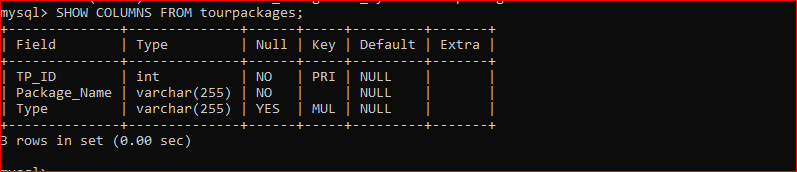
**9. Payment\_transaction:**



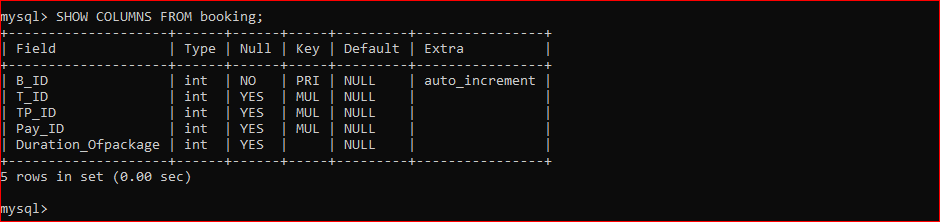
**10. Packageprice:**



**11. Tourpackage:**

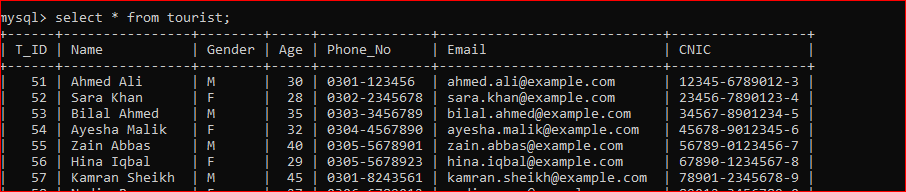


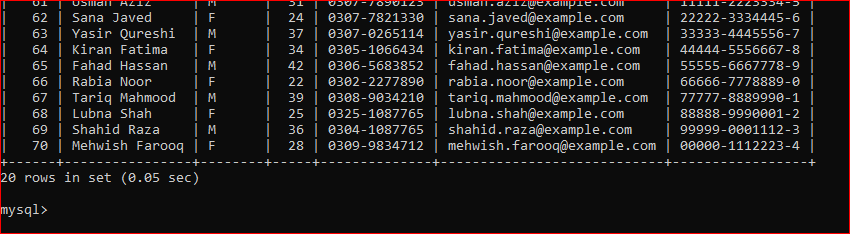
**12. Booking:**



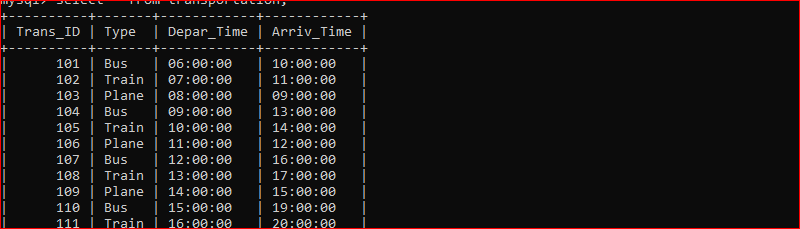
**4.2 DATA SAMPLES INSIDE TABLES:**

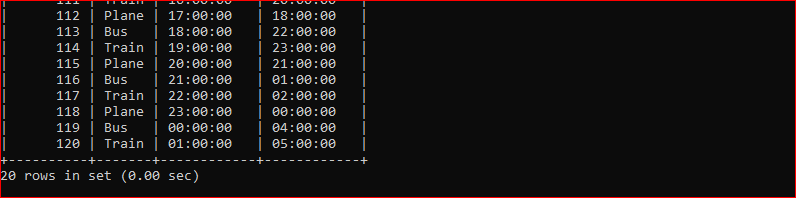
1. **Tourist:**



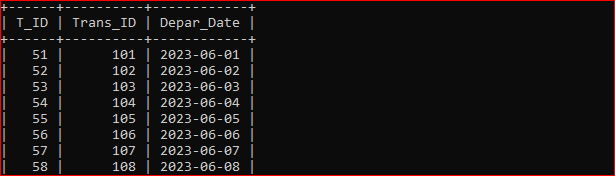


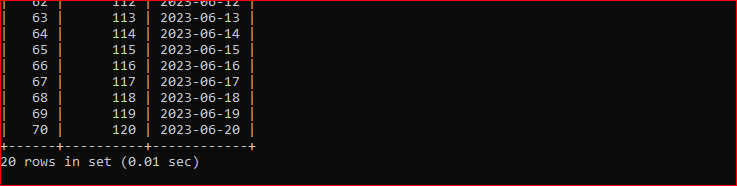
1. **Transportation:**



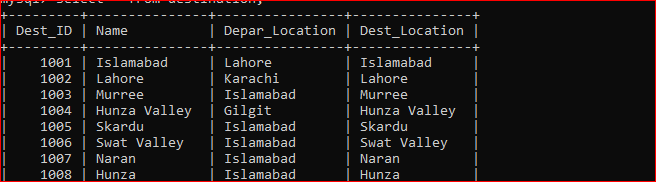


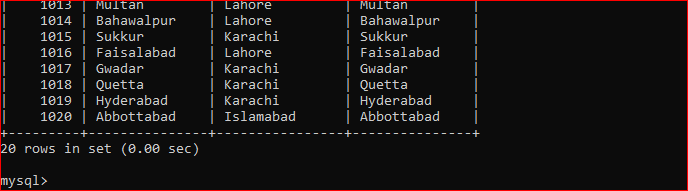
1. **Journey:**



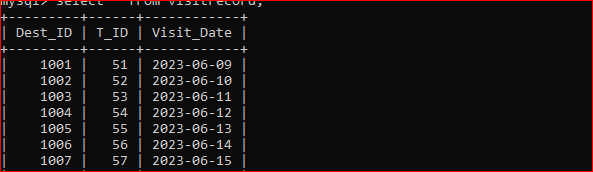


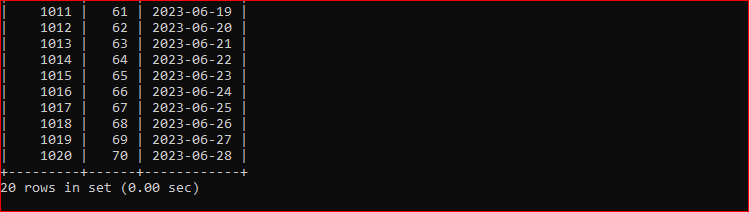
1. **Destination:**



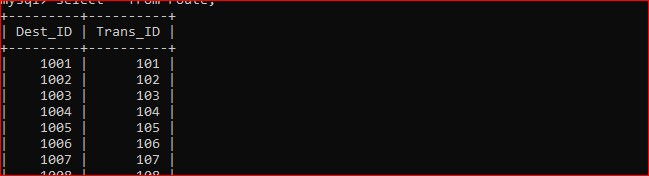


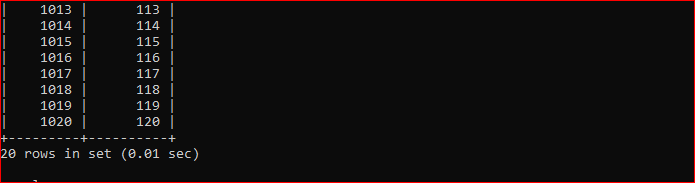
1. **Visitrecord:**



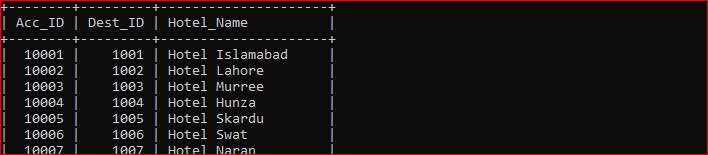


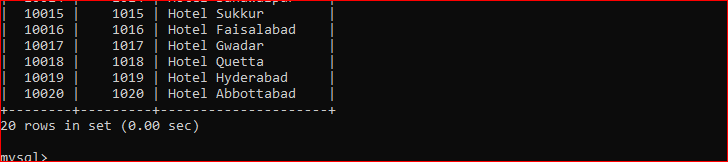
1. **Route:**



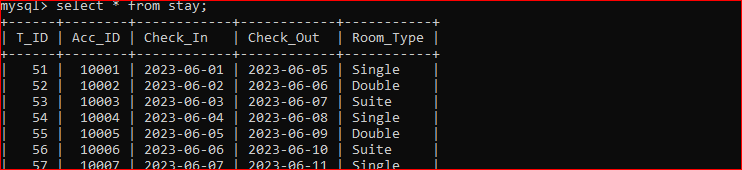


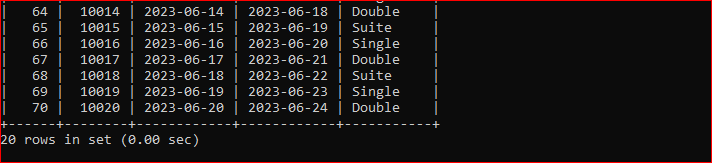
1. **Accomodation:**



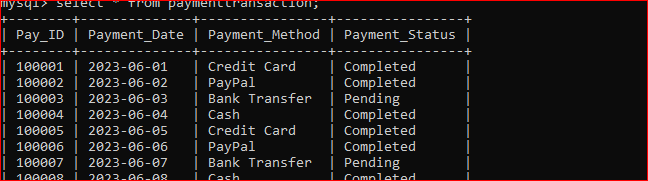


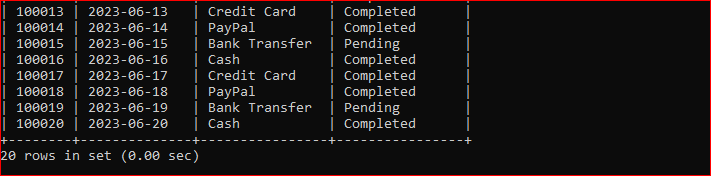
1. **Stay:**





1. **Payment\_transaction:**

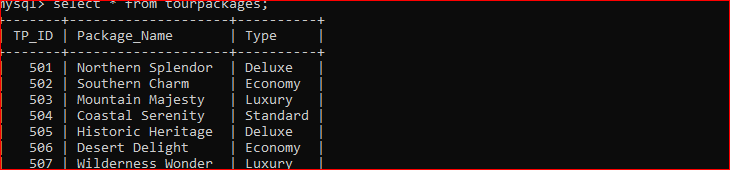


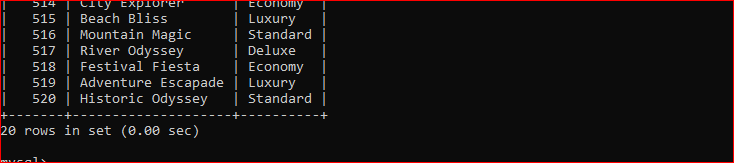


1. **Packageprice:**

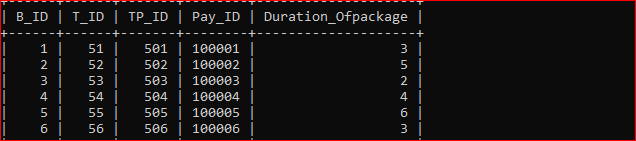


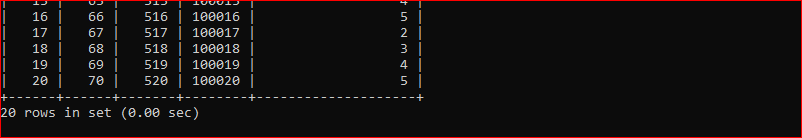
1. **Tourpackage:**





1. **Booking:**





**4.3 QUERIES RESULTS:**

|  |  |
| --- | --- |
| **01** | Count of total tourists according to gender. |
| **OUTPUT** |  |
| **02** | Find the transportation details for tourists departing on a specific date. |
| **OUTPUT** |  |
| **03** | List all tourists along with their booked packages and the duration of each package. |
| **OUTPUT** |  |
| **04** | Retrieve the booking details (including tourist name, package name, and duration) for tourists over a certain age. |
| **OUTPUT** |  |
| **05** | List all transportations available for a specific destination. |
| **OUTPUT** |  |
| **06** | Get the details of tourists who have made payments via a ‘Credit Card’. |
| **OUTPUT** |  |
| **07** | Total number of bookings per package, ordered by total bookings. |
| **OUTPUT** |  |
| **08** | Total duration of packages booked by tourists, ordered by total duration. |
| **OUTPUT** |  |
| **09** | Total amount paid by payment method, having more than 3 transactions, ordered by the total amount paid. |
| **OUTPUT** |  |
| **10** | Total number of journeys per transportation type, having more than 2 journeys, ordered by total journeys. |
| **OUTPUT** |  |
| **11** | Find all tourists who have stayed in a hotel located in 'Islamabad'. |
| **OUTPUT** |  |
| **12** | Retrieve the names and email addresses of tourists who visited 'Naran'. |
| **OUTPUT** |  |

# CHAPTER 5: Interface Design

## LANGUAGE/FRAMEWORK:

**Chosen Technology:**

* **Programming Language:** Python
* **GUI Framework:** Tkinter
* **Database Connector:** mysql-connector-python

**Reasons for Selection:**

* **Python:**
  + **Ease of Learning and Use:** Python's simple syntax and readability accelerate development and reduce the learning curve.
  + **Extensive Libraries:** Python offers a wide range of libraries and frameworks, making it versatile for various applications.
* **Tkinter:**
  + **Standard Library:** Tkinter is included with Python, eliminating the need for additional installations and simplifying the setup process.
  + **Ease of Use:** Tkinter provides a straightforward way to create GUI applications.
  + **Documentation and Community Support:** Tkinter has extensive documentation and a strong community for troubleshooting and expanding functionality.
* **mysql-connector-python:**
  + **Compatibility:** This library is designed to work seamlessly with MySQL, ensuring smooth database operations.
  + **Ease of Use:** It offers a simple API for connecting to MySQL databases, executing queries, and handling results.
  + **Robust Error Handling:** It includes built-in mechanisms for handling common database errors, maintaining application stability.

**Key Features and Their Contributions:**

* **Tkinter:**
  + **Widgets:** Tkinter provides a wide range of widgets (buttons, labels, text boxes, etc.) essential for creating interactive GUI elements.
  + **Customization:** The ability to customize the application's look and feel with themes and styles.
  + **Event Handling:** Tkinter supports event-driven programming, allowing for responsive user interfaces.
* **mysql-connector-python:**
  + **Secure Connection Handling:** Provides secure methods for connecting to and interacting with MySQL databases.
  + **SQL Execution:** Facilitates the execution of SQL queries directly from Python, allowing for dynamic data manipulation.
  1. **DATABASE CONNECTIVITY:**

The GUI client connects to the MySQL database using the mysql-connector-python library. This library provides a straightforward way to establish a connection, execute SQL commands, and handle results.

**Code Snippet for Database Connection:**

python

Copy code

import mysql.connector

from tkinter import messagebox

# Connect to the MySQL database

def connect\_to\_db():

try:

return mysql.connector.connect(

host="localhost",

user="root",

password="Jalal@21",

database="Tourism\_Management\_System"

)

except mysql.connector.Error as err:

messagebox.showerror("Error", f"Error: {err}")

return None

**Explanation:**

* **Connection Parameters:** The connect method takes parameters such as host, user, password, and database to establish a connection to the MySQL database.
* **Error Handling:** A try-except block catches and displays any connection errors using a message box, ensuring the user is informed if the connection fails.
  1. **STORED PROCEDURES AND FUNCTIONS:**

**Stored Procedures:**

DELIMITER //

CREATE PROCEDURE InsertTourist(

IN p\_T\_ID INT,

IN p\_Name VARCHAR(255),

IN p\_Gender CHAR(1),

IN p\_Age INT,

IN p\_Phone\_No VARCHAR(255),

IN p\_Email VARCHAR(255),

IN p\_CNIC VARCHAR(20)

)

BEGIN

INSERT INTO Tourist (T\_ID, Name, Gender, Age, Phone\_No, Email, CNIC)

VALUES (p\_T\_ID, p\_Name, p\_Gender, p\_Age, p\_Phone\_No, p\_Email, p\_CNIC);

END //

CREATE PROCEDURE UpdateTourist(

IN p\_T\_ID INT,

IN p\_Name VARCHAR(255),

IN p\_Gender CHAR(1),

IN p\_Age INT,

IN p\_Phone\_No VARCHAR(255),

IN p\_Email VARCHAR(255),

IN p\_CNIC VARCHAR(20)

)

BEGIN

UPDATE Tourist

SET Name = p\_Name, Gender = p\_Gender, Age = p\_Age, Phone\_No = p\_Phone\_No, Email = p\_Email, CNIC = p\_CNIC

WHERE T\_ID = p\_T\_ID;

END //

CREATE PROCEDURE DeleteTourist(

IN p\_T\_ID INT

)

BEGIN

DELETE FROM Tourist WHERE T\_ID = p\_T\_ID;

END //

CREATE PROCEDURE SearchTourist(

IN p\_T\_ID INT

)

BEGIN

SELECT \* FROM Tourist WHERE T\_ID = p\_T\_ID;

END //

CREATE PROCEDURE GetAllTourists()

BEGIN

SELECT \* FROM Tourist;

END //

DELIMITER ;

**Explanation:**

* **InsertTourist:**
  + **Objective:** Adds a new tourist record.
  + **Implementation:** Takes tourist details as input parameters and inserts them into the Tourist table.
  + **Enhancement:** Simplifies adding new tourist records.
* **UpdateTourist:**
  + **Objective:** Updates an existing tourist record.
  + **Implementation:** Takes updated details and the tourist ID to modify the corresponding record.
  + **Enhancement:** Allows modifying existing records efficiently.
* **DeleteTourist:**
  + **Objective:** Deletes a tourist record.
  + **Implementation:** Takes the tourist ID as input and deletes the corresponding record.
  + **Enhancement:** Facilitates removing unwanted records.
* **SearchTourist:**
  + **Objective:** Searches for a specific tourist record.
  + **Implementation:** Takes the tourist ID and retrieves the corresponding record.
  + **Enhancement:** Enables quick lookup of specific records.
* **GetAllTourists:**
  + **Objective:** Retrieves all tourist records.
  + **Implementation:** Selects all records from the Tourist table.
  + **Enhancement:** Provides an overview of all tourist records.
* **Functions:**

Within the Python code, several functions are defined to facilitate database operations and GUI interactions. Here are the key functions:

* **connect\_to\_db()**: Establishes a connection to the MySQL database.
* **get\_fields\_for\_table(table)**: Returns the columns for a given table.
* **load\_data()**: Loads all data from the current table into the Treeview widget.
* **add\_data()**: Adds a new record to the current table.
* **modify\_data()**: Modifies an existing record in the current table.
* **delete\_data()**: Deletes a record from the current table.
* **search\_data()**: Searches for a record in the current table.
* **show\_all\_data()**: Displays all records from the current table in a Treeview widget.

These functions ensure smooth interactions between the GUI and the database, providing a seamless user experience.

4o

please answer the 5.3 completely with functions

ChatGPT

* **5.3. STORED PROCEDURES AND FUNCTIONS:**

**Stored Procedures:**

sql

Copy code

DELIMITER //

CREATE PROCEDURE InsertTourist(

IN p\_T\_ID INT,

IN p\_Name VARCHAR(255),

IN p\_Gender CHAR(1),

IN p\_Age INT,

IN p\_Phone\_No VARCHAR(255),

IN p\_Email VARCHAR(255),

IN p\_CNIC VARCHAR(20)

)

BEGIN

INSERT INTO Tourist (T\_ID, Name, Gender, Age, Phone\_No, Email, CNIC)

VALUES (p\_T\_ID, p\_Name, p\_Gender, p\_Age, p\_Phone\_No, p\_Email, p\_CNIC);

END //

CREATE PROCEDURE UpdateTourist(

IN p\_T\_ID INT,

IN p\_Name VARCHAR(255),

IN p\_Gender CHAR(1),

IN p\_Age INT,

IN p\_Phone\_No VARCHAR(255),

IN p\_Email VARCHAR(255),

IN p\_CNIC VARCHAR(20)

)

BEGIN

UPDATE Tourist

SET Name = p\_Name, Gender = p\_Gender, Age = p\_Age, Phone\_No = p\_Phone\_No, Email = p\_Email, CNIC = p\_CNIC

WHERE T\_ID = p\_T\_ID;

END //

CREATE PROCEDURE DeleteTourist(

IN p\_T\_ID INT

)

BEGIN

DELETE FROM Tourist WHERE T\_ID = p\_T\_ID;

END //

CREATE PROCEDURE SearchTourist(

IN p\_T\_ID INT

)

BEGIN

SELECT \* FROM Tourist WHERE T\_ID = p\_T\_ID;

END //

CREATE PROCEDURE GetAllTourists()

BEGIN

SELECT \* FROM Tourist;

END //

DELIMITER ;

**Explanation:**

* **InsertTourist:**
  + **Objective:** Adds a new tourist record.
  + **Implementation:** Takes tourist details as input parameters and inserts them into the Tourist table.
  + **Enhancement:** Simplifies adding new tourist records.
* **UpdateTourist:**
  + **Objective:** Updates an existing tourist record.
  + **Implementation:** Takes updated details and the tourist ID to modify the corresponding record.
  + **Enhancement:** Allows modifying existing records efficiently.
* **DeleteTourist:**
  + **Objective:** Deletes a tourist record.
  + **Implementation:** Takes the tourist ID as input and deletes the corresponding record.
  + **Enhancement:** Facilitates removing unwanted records.
* **SearchTourist:**
  + **Objective:** Searches for a specific tourist record.
  + **Implementation:** Takes the tourist ID and retrieves the corresponding record.
  + **Enhancement:** Enables quick lookup of specific records.
* **GetAllTourists:**
  + **Objective:** Retrieves all tourist records.
  + **Implementation:** Selects all records from the Tourist table.
  + **Enhancement:** Provides an overview of all tourist records.

**Functions in the Python Code:**

The following functions are defined in the Python code to facilitate database operations and GUI interactions:

* **connect\_to\_db()**:
  + **Objective:** Establishes a connection to the MySQL database.
  + **Implementation:** Uses the mysql.connector library to connect to the database. Includes error handling to display error messages if the connection fails.

python

Copy code

def connect\_to\_db():

try:

return mysql.connector.connect(

host="localhost",

user="root",

password="Jalal@21",

database="Tourism\_Management\_System"

)

except mysql.connector.Error as err:

messagebox.showerror("Error", f"Error: {err}")

return None

* **get\_fields\_for\_table(table)**:
  + **Objective:** Returns the columns for a given table.
  + **Implementation:** Provides a mapping of table names to their respective columns.

python

Copy code

def get\_fields\_for\_table(self, table):

if table == "Tourist":

return ["T\_ID", "Name", "Gender", "Age", "Phone\_No", "Email", "CNIC"]

elif table == "Transportation":

return ["Trans\_ID", "Type", "Depar\_Time", "Arriv\_Time"]

elif table == "Journey":

return ["T\_ID", "Trans\_ID", "Depar\_Date"]

elif table == "Destination":

return ["Dest\_ID", "Name", "Depar\_Location", "Dest\_Location"]

elif table == "VisitRecord":

return ["Dest\_ID", "T\_ID", "Visit\_Date"]

elif table == "Route":

return ["Dest\_ID", "Trans\_ID"]

elif table == "Accomodation":

return ["Acc\_ID", "Dest\_ID", "Hotel\_Name"]

elif table == "Stay":

return ["T\_ID", "Acc\_ID", "Check\_In", "Check\_Out", "Room\_Type"]

elif table == "PaymentTransaction":

return ["Pay\_ID", "Payment\_Date", "Payment\_Method", "Payment\_Status"]

elif table == "PackagePrice":

return ["Type", "Price"]

elif table == "TourPackages":

return ["TP\_ID", "Package\_Name", "Type"]

elif table == "Booking":

return ["B\_ID", "T\_ID", "TP\_ID", "Pay\_ID", "Duration\_Ofpackage"]

* **load\_data()**:
  + **Objective:** Loads all data from the current table into the Treeview widget.
  + **Implementation:** Connects to the database, retrieves all records from the current table, and populates the Treeview widget.

python

Copy code

def load\_data(self):

table = self.current\_table

if table:

self.treeview.delete(\*self.treeview.get\_children())

conn = connect\_to\_db()

if conn:

cursor = conn.cursor()

cursor.execute(f"SELECT \* FROM {table}")

for row in cursor.fetchall():

self.treeview.insert("", tk.END, values=row)

cursor.close()

conn.close()

* **add\_data()**:
  + **Objective:** Adds a new record to the current table.
  + **Implementation:** Collects data from entry fields, connects to the database, and inserts the new record. Calls the show\_all\_data() function to refresh the display.

python

Copy code

def add\_data(self):

table = self.current\_table

fields = self.get\_fields\_for\_table(table)

values = [entry.get() for entry in self.field\_entries]

if all(values):

conn = connect\_to\_db()

if conn:

cursor = conn.cursor()

placeholders = ", ".join(["%s"] \* len(fields))

try:

cursor.execute(f"INSERT INTO {table} VALUES ({placeholders})", values)

conn.commit()

cursor.close()

conn.close()

self.show\_all\_data()

self.clear\_entries()

except mysql.connector.Error as err:

messagebox.showerror("Error", f"Error: {err}")

cursor.close()

conn.close()

else:

messagebox.showerror("Error", "All fields are required")

* **modify\_data()**:
  + **Objective:** Modifies an existing record in the current table.
  + **Implementation:** Collects updated data from entry fields, connects to the database, and updates the record. Calls the show\_all\_data() function to refresh the display.

python

Copy code

def modify\_data(self):

if self.treeview is None:

messagebox.showerror("Error", "No data to modify")

return

table = self.current\_table

fields = self.get\_fields\_for\_table(table)

values = [entry.get() for entry in self.field\_entries]

if all(values):

selected\_item = self.treeview.selection()

if selected\_item:

item = self.treeview.item(selected\_item)

id\_field = fields[0]

id\_value = item['values'][0]

conn = connect\_to\_db()

if conn:

cursor = conn.cursor()

set\_clause = ", ".join([f"{field}=%s" for field in fields[1:]])

cursor.execute(f"UPDATE {table} SET {set\_clause} WHERE {id\_field}=%s", values[1:] + [id\_value])

conn.commit()

cursor.close()

conn.close()

self.show\_all\_data()

self.clear\_entries()

else:

messagebox.showerror("Error", "No item selected for modification")

else:

messagebox.showerror("Error", "All fields are required")

* **delete\_data()**:
  + **Objective:** Deletes a record from the current table.
  + **Implementation:** Connects to the database and deletes the selected record. Calls the show\_all\_data() function to refresh the display.

python

Copy code

def delete\_data(self):

if self.treeview is None:

messagebox.showerror("Error", "No data to delete")

return

table = self.current\_table

fields = self.get\_fields\_for\_table(table)

selected\_item = self.treeview.selection()

if selected\_item:

item = self.treeview.item(selected\_item)

id\_field = fields[0]

id\_value = item['values'][0]

conn = connect\_to\_db()

if conn:

cursor = conn.cursor()

try:

cursor.execute(f"DELETE FROM {table} WHERE {id\_field}=%s", (id\_value,))

conn.commit()

cursor.close()

conn.close()

self.show\_all\_data()

except mysql.connector.Error as err:

messagebox.showerror("Error", f"Error: {err}")

cursor.close()

conn.close()

else:

messagebox.showerror("Error", "No item selected for deletion")

* **search\_data()**:
  + **Objective:** Searches for a record in the current table.
  + **Implementation:** Connects to the database and retrieves the record matching the search criteria.

python

Copy code

def search\_data(self):

table = self.current\_table

fields = self.get\_fields\_for\_table(table)

search\_field = self.search\_field.get()

search\_value = self.search\_entry.get()

if search\_field and search\_value:

conn = connect\_to\_db()

if conn:

cursor = conn.cursor()

query = f"SELECT \* FROM {table} WHERE {search\_field} LIKE %s"

cursor.execute(query, (f"%{search\_value}%",))

self.treeview.delete(\*self.treeview.get\_children())

for row in cursor.fetchall():

self.treeview.insert("", tk.END, values=row)

cursor.close()

conn.close()

else:

messagebox.showerror("Error", "Please select a field and enter a search value")

* **show\_all\_data()**:
  + **Objective:** Displays all records from the current table in a Treeview widget.
  + **Implementation:** Connects to the database and retrieves all records from the current table. Populates the Treeview widget with the retrieved data.

python

Copy code

def show\_all\_data(self):

table = self.current\_table

self.treeview.delete(\*self.treeview.get\_children())

conn = connect\_to\_db()

if conn:

cursor = conn.cursor()

cursor.execute(f"SELECT \* FROM {table}")

for row in cursor.fetchall():

self.treeview.insert("", tk.END, values=row)

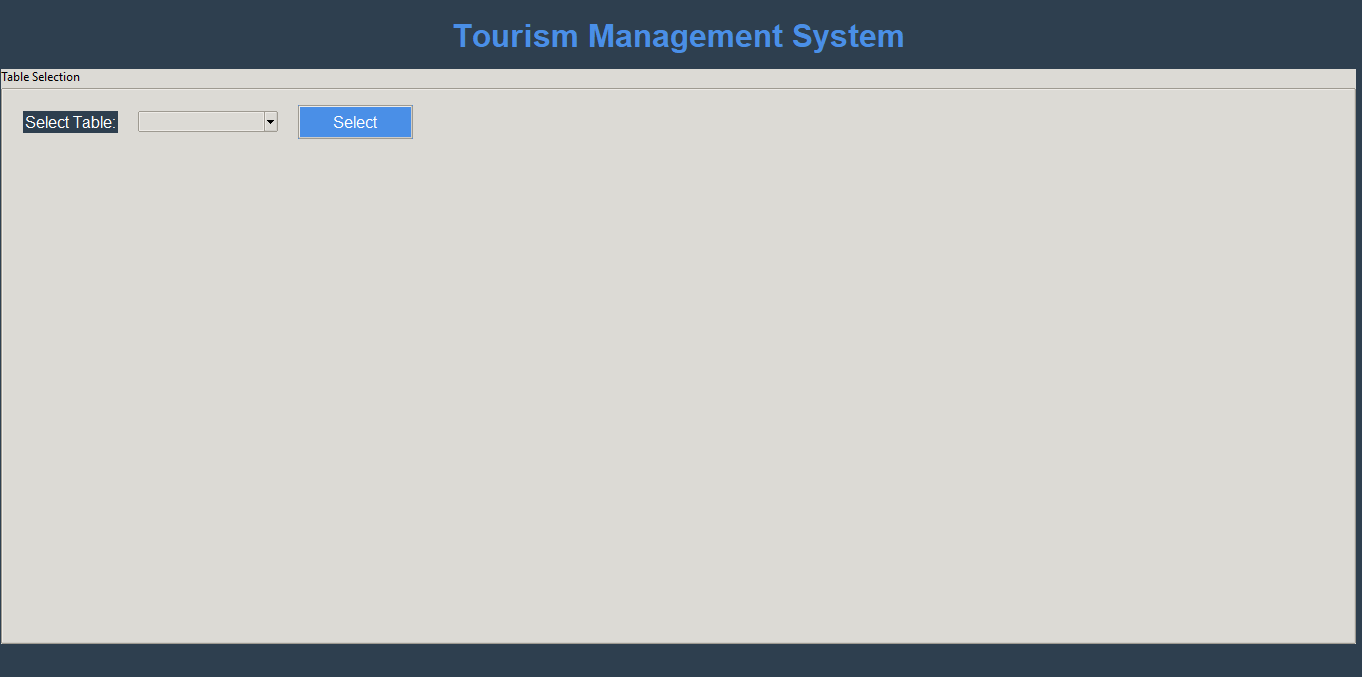
cursor.close()

conn.close()

These functions ensure smooth interactions between the GUI and the database, providing a seamless user experience for managing tourist records.

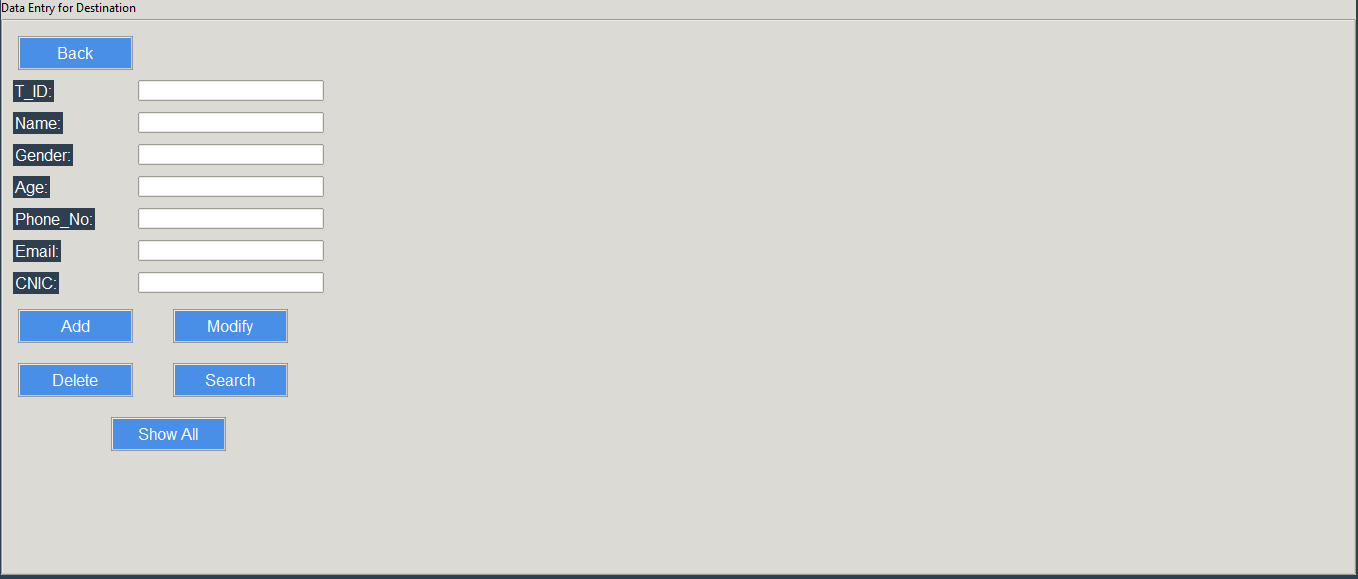
* 1. **INTERFACES:**

**Main Interface:**

****

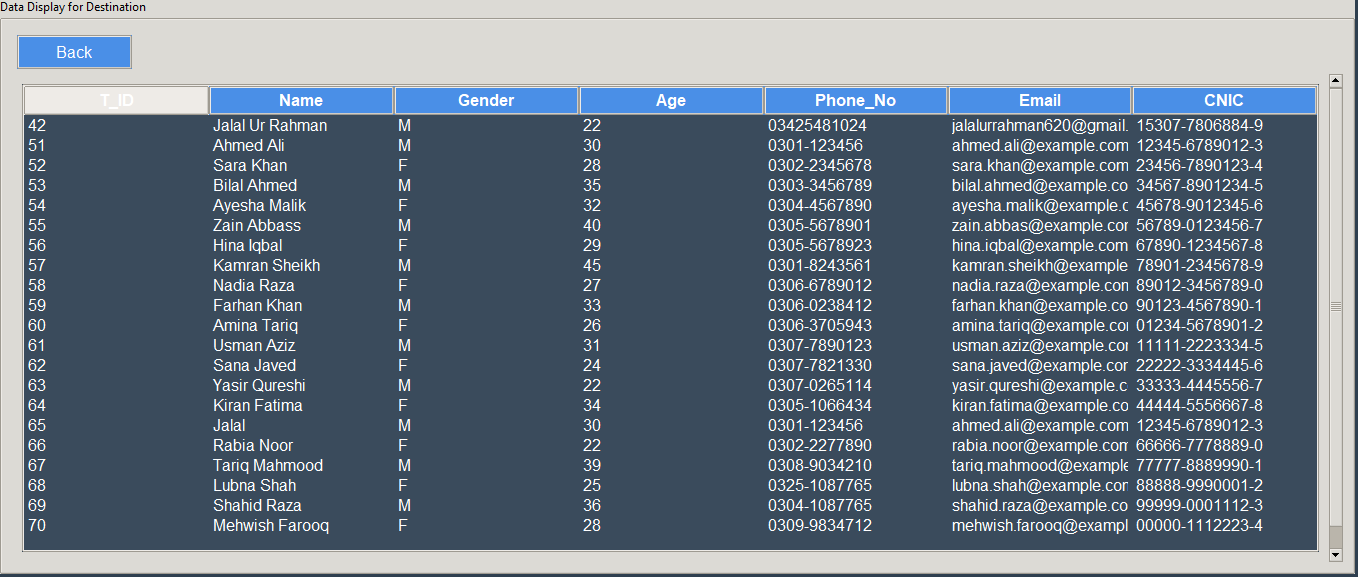
**Description:** The main window of the application where users can select tables and perform operations.

**Data Entry Form:**

****

**Description:** A form for entering data into the selected table. It includes fields for each column in the table and buttons for CRUD operations.

**Data Display:**

****

**Description:** A display frame showing all records from the selected table in a Treeview widget. Includes a scrollbar for navigation.

**Functionality Descriptions:**

* **Table Selection:** Users can select which table they want to interact with.
* **CRUD Operations:** Users can add, modify, delete, and search records within the selected table.
* **Data Display:** Users can view all records from the selected table in a tabular format.

These interfaces provide a user-friendly way to manage tourism-related data, ensuring that all necessary operations can be performed efficiently.

# 6: CONCLUSION

## LESSONS LEARNED:

During the tourism management system project using MySQL, we focused on clear and detailed requirements documentation, using use case and ER diagrams to understand the project's scope. We created a project plan with milestones and used Gantt charts and task management tools to ensure timely delivery. Identifying and monitoring risks early allowed us to develop effective mitigation strategies. Quality assurance was achieved through thorough testing at various stages, using automated tools for efficiency. Teamwork was strengthened through regular communication, clear role allocation, constructive conflict resolution, and peer learning. Adaptability to changes in requirements or technology, along with thorough documentation, was crucial for smooth project progression

*.***6.2. CHALLENGES AND SOLUTIONS:**

Throughout the project, several challenges were encountered: ensuring data integrity across a complex database structure, managing delimiter functions effectively in SQL queries and procedures, designing a user-friendly GUI interface that met diverse user needs, and establishing a secure connection between the website front end and MySQL backend. To address these challenges, we prioritized database normalization and strict data validation for data integrity, implemented structured SQL coding practices for delimiter functions, and took help from our seniors in the connection of the backend with the front end.

**6.3. FUTURE WORK AND IMPROVEMENTS:**

Looking ahead, improvements to the tourism management system will focus on enhancing user experience through a streamlined GUI for easier navigation and accessibility, alongside the development of a mobile-friendly interface. Advanced features will include robust reporting capabilities with customizable analytics and the integration of AI for predictive insights. To enhance scalability and performance, the system will consider migrating to cloud-based infrastructure and continually optimizing SQL queries and server setups. Strengthened security measures, including enhanced encryption and regular audits, will ensure data protection. Integration with APIs and IoT will broaden compatibility with other systems, while implementing CRM functionalities and providing comprehensive user support will boost engagement. Compliance with regulatory standards like GDPR and initiatives for sustainable tourism practices will further underscore reliability and user confidence in the system.

**6.4. FINAL THOUGHTS:**

As a fourth-semester student reflecting on the tourism management system project, it has been a rewarding learning experience. I've gained valuable insights into database design, project planning, and the importance of user-centric interfaces. Overcoming challenges like ensuring data integrity and optimizing performance has been both educational and fulfilling. I appreciate the opportunity to work collaboratively with my team, learning the importance of effective communication and teamwork in achieving project goals. Looking forward, I'm excited about applying these skills to future projects and continuing to explore new technologies to enhance user experiences and system efficiency.

# REFERENCES

|  |  |
| --- | --- |
| [1] | "prestinetravels," [Online]. Available: https://prestinetravels.com/trip/best-of-swat-kalam-malamjabba-pakistan-tour/#wte-booking-datetime. |
| [2] | "boardmix," [Online]. Available: https://boardmix.com/app/editor/45jBxCPzK7eX1rby1aD7eQ. |
| [3] | "blizin," [Online]. Available: https://www.blizin.com/kalash-valley-tour-packages. |