Kishkinda University

Ballari

INTERNSHIP

Report On

TRAVEL HISTORY TRACKER

Submitted in partial fulfillment of the requirements for the award of degree of

Bachelor of Engineering
In

Computer Science and Engineering

Submitted by

Internship Carried Out By

EZ TRAININGS & TECHNOLOGIES PVT.LTD HYDERABAD

Internal Guide **External Guide**

DECLARATION

I, second year student of Computer Science and Engineering Of **KISHKINDA UNIVERSITY**, Ballari, declare that Internship entitled in is a part of internship Training successfully carried out by **EZ TECHNOLOGIES & TRAININGS PVT.LTD**, **HYDERABAD** at Kishkinda University. This report is submitted in partial fulfillment of requirements for the award of the degree, Bachelor of Engineering in Computer Science and Engineering.

Date: 28/09/2024
Place: Ballari **Signature of the Student**

Our Code :

```
import mysql.connector as mysql

db = mysql.connect(
    host='localhost',
    user='root',
    password='user',
    database='TravelRecord'
)
mycursor=db.cursor()
class TravelRecord:
    def _init_(self, history_id, individual_name, destination,
travel_date=None, return_date=None):
        self.history_id = history_id
        self.individual_name = individual_name
        self.destination = destination
        self.travel_date = travel_date
        self.return_date = return_date

    def _repr_(self):
        return (f"TravelRecord(history_id={self.history_id}, "
                f"individual_name='{self.individual_name}', "
                f"destination='{self.destination}', "
                f"travel_date='{self.travel_date}', "
                f"return_date='{self.return_date}')")

class TravelHistoryTracker:
    def createdb():
        try:
            mycursor.execute("CREEATED DATABASE RECORD")
        except Exception:
            print('Already created db')
    def useDB():
        try:
            mycursor.execute("USE RECORD")
            print("using DB")
        except Exception:
            print('Already use DB')
```

```
    def create_record(self, history_id, individual_name, destination,
travel_date=None, return_date=None):
        cursor = self.db_connection.cursor()
        try:
            cursor.execute("""
                INSERT INTO travel_records (history_id, individual_name,
destination, travel_date, return_date)
                VALUES (%s, %s, %s, %s, %s)
            """, (history_id, individual_name, destination, travel_date,
return_date))
            self.db_connection.commit()
            print('created success...')
        except mysql.Error as err:
            print(f"Error: {err}")
        else:
            print('created already')
        finally:
            cursor.close()

    def read_record(self, history_id):
        cursor = self.db_connection.cursor()
        cursor.execute("SELECT * FROM travel_records WHERE history_id =
%s", (history_id,))
        record = cursor.fetchone()
        cursor.close()
        if record:
            return TravelRecord(*record)
        return "Record not found."

    def update_record(self, history_id, individual_name=None,
destination=None, travel_date=None, return_date=None):
        cursor = self.db_connection.cursor()
        try:
            updates = []
            params = []

            if individual_name:
                updates.append("individual_name = %s")
```

```
                params.append(individual_name)
            if destination:
                updates.append("destination = %s")
                params.append(destination)
            if travel_date:
                updates.append("travel_date = %s")
                params.append(travel_date)
            if return_date:
                updates.append("return_date = %s")
                params.append(return_date)

            if updates:
                sql = f"UPDATE travel_records SET {', '.join(updates)}
WHERE history_id = %s"
                params.append(history_id)
                cursor.execute(sql, tuple(params))
                self.db_connection.commit()
            else:
                return "No updates provided."
        except mysql.Error as err:
            print(f"Error: {err}")
        finally:
            cursor.close()

    def delete_record(self, history_id):
        cursor = self.db_connection.cursor()
        try:
            cursor.execute("DELETE FROM travel_records WHERE history_id =
%s", (history_id,))
            self.db_connection.commit()
            if cursor.rowcount == 0:
                return "Record not found."
        except mysql.Error as err:
            print(f"Error: {err}")
        finally:
            cursor.close()

    def analyze_travel_behaviors(self):
        cursor = self.db_connection.cursor()
```

```
        cursor.execute("SELECT destination, COUNT(*) FROM travel_records
GROUP BY destination")
        result = cursor.fetchall()
        cursor.close()

        destination_count = {destination: count for destination, count in
result}
        return destination_count

# Create an instance of TravelHistoryTracker
tracker = TravelHistoryTracker()
tracker.useDB()
# Create a record
#tracker.create_record(91, 'Rajesh', 'Japan', '2023-01-01', '2024-02-10')


# Read a record
#record = tracker.read_record(99)
#print(record)

# Update a record
#tracker.update_record(99, destination='Canada')

# Read the updated record
#updated_record = tracker.read_record(99)
#print(updated_record)

# Delete a record
tracker.delete_record(99)


# Analyze travel behaviors
pattern = tracker.analyze_travel_behaviors()
print(pattern)
```

PRESENTATION ON TRAVELS RECORD

BATCH NO :- 5

MEMBER OF BATCH NO :- 5

1) RAGHAVENDRA Y
2) RAJESH
3) AKASH B
4) MD GHOUSE
5) MD FAISAL

INTRODUCTION

- This Python script leverages MySQL as a backend database to manage travel records efficiently. It provides a comprehensive system for creating, reading, updating, deleting, and analyzing travel records through a class-based approach.

- The script connects to a MySQL database and allows users to interact with travel data, storing information such as traveler's name, destination, and travel dates. A key feature of the script is its ability to analyze travel behaviors, giving insights into the frequency of visits to different destinations. This serves as a useful tool for travel agencies or businesses that track customer travel habits.

DATABASE CONNECTION

- The code begins by establishing a connection to a MySQL database named TravelRecord using the mysql.connector library.

TRAVELRECORD CLASS :

- This class represents a travel record.
- It has attributes: history_id, individual_name, destination, travel_date, and return_date.
- The_repr_ method provides a string representation for debugging.

Travel History Tracker Class

- Methods:
-
- create_record: Inserts a new travel record into the database.
-
- Read_record: Fetches a travel record by history_id.
-
- Update_record: Updates specified fields of a travel record.
-
- Delete_record: Deletes a record by history_id
-
- analyze_travel_behaviors: Analyzes travel patterns by counting how many times each destination was visited.

USAGE OF TRAVEL HISTORY TRAKER

- An instance of TravelHistory Tracker is created using the database connection.
- Several operations are performed:
- Creating a new travel record.
- Reading an existing record.
- Updating a record,
- Deleting a record.
- travel behaviors.

CONCLUSION

- This script efficiently manages travel data by providing CRUD (Create, Read, Update, Delete) functionalities through a MySQL connection. With the additional feature of analyzing travel patterns, the system offers valuable insights into travel behaviors across various destinations. The flexibility of updating and retrieving data ensures that the script can be applied in real-world scenarios where accurate and dynamic travel data management is essential.

THANK YOU