ADVANCES IN DATABASES

Spatial Database Applications: Develop a Spatial Database Application and Perform Spatial Query Operations

Mahima S

[2022115018]

Introduction

The Spatial Database Application presented here is designed to leverage PostgreSQL with PostGIS extensions for performing spatial query operations. This application provides a user-friendly interface to interact with spatial data and offers functionalities such as identifying landmarks within a city, calculating distances, retrieving visitor data, and more. It serves as a practical example of utilizing spatial databases in real-world scenarios.

Objectives

The primary objectives of this application are:

1. To develop a spatial database application that demonstrates the use of spatial query operations.
2. To enable efficient querying and analysis of spatial data.
3. To provide insights into the integration of spatial databases with Python for application development.

Features and Functionalities

1. Find Landmarks in a City

This feature allows users to retrieve a list of landmarks within a specified city by querying the database.

- **Query Used:**

```
SELECT l.name

FROM landmarks l

JOIN cities c ON l.city_id = c.id

WHERE c.name = %s;
```

2. Find Landmarks Within a Radius

Users can search for landmarks within a given radius from a city center.

- **Query Used:**

```
SELECT l.name
FROM landmarks l
JOIN cities c ON l.city_id = c.id
WHERE c.name = %s AND ST_DWithin(c.location, l.location, %s);
```

3. Calculate Distance Between Two Landmarks

The application calculates the distance between two landmarks using PostGIS spatial functions.

- **Query Used:**

```
SELECT ST_Distance(
   (SELECT location FROM landmarks WHERE name = %s),
   (SELECT location FROM landmarks WHERE name = %s)
) AS distance_in_meters;
```

4. Retrieve Visitors to a Landmark

Users can fetch the details of visitors to a specific landmark, including their names and visit dates.

- **Query Used:**

```
SELECT v.name, v.visit_date
FROM visitors v
JOIN landmarks l ON v.landmark_id = l.id
WHERE l.name = %s;
```

5. Fetch Reviews for a Landmark

This functionality retrieves reviews, ratings, and review dates for a selected landmark.

- **Query Used:**

```
SELECT r.review_text, r.rating, r.review_date
FROM reviews r
JOIN landmarks l ON r.landmark_id = l.id
WHERE l.name = %s;
```

6. Show Top 5 Most Visited Landmarks

The application identifies the most visited landmarks by counting visitor entries.

- **Query Used:**

```
SELECT l.name, COUNT(v.id) AS visit_count
FROM landmarks l
LEFT JOIN visitors v ON l.id = v.landmark_id
GROUP BY l.id
ORDER BY visit_count DESC
LIMIT 5;
```

7. Calculate Average Rating for a Landmark

Users can calculate the average rating of a landmark based on its reviews.

- **Query Used:**

```
SELECT AVG(r.rating) AS average_rating
FROM reviews r
JOIN landmarks l ON r.landmark_id = l.id
WHERE l.name = %s;
```

8. Identify Landmarks with No Visitors

The application highlights landmarks that have not been visited by anyone.

- **Query Used:**

```
SELECT l.name
FROM landmarks l
LEFT JOIN visitors v ON l.id = v.landmark_id
WHERE v.id IS NULL;
```

Implementation

Technologies Used

- **Database:** PostgreSQL with PostGIS extension
- **Programming Language:** Python
- **Libraries:**
    - psycopg2 for PostgreSQL connection
    - colorama for colored console output
    - logging for tracking application activity
    - time for simulating progress indicators

Application Design

The application is menu-driven and provides users with an interactive console interface. Key components include:

1. **Database Connection:** A reusable function ensures secure and consistent connections to the database.
2. **User Interface:** Menu options guide users through the available features.
3. **Error Handling:** Comprehensive error handling ensures graceful recovery from database or query-related issues.

Menu Interface:

Menu:

1. Find landmarks in a city

2. Find landmarks within a radius

3. Calculate distance between two landmarks

4. Find visitors to a landmark

5. Fetch reviews for a landmark

6. Show top 5 most visited landmarks

7. Show average rating for a landmark

8. Show landmarks with no visitors

9. Exit

Execution Flow

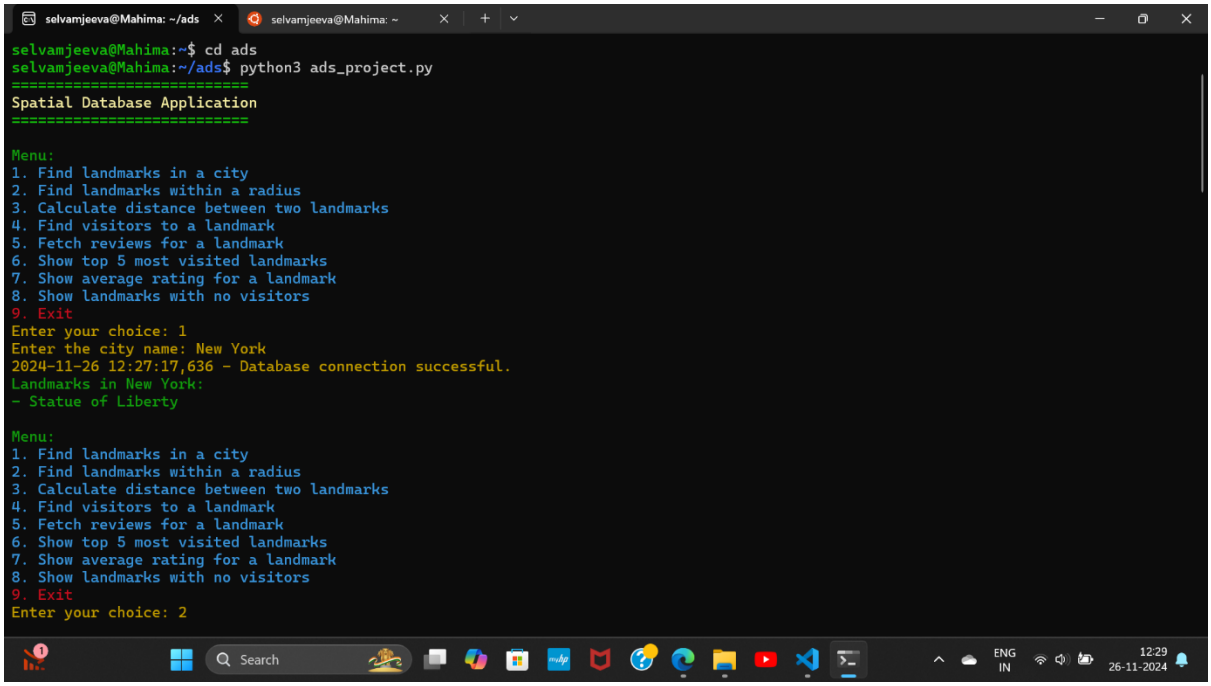
1. **Users select an option from the menu.**
2. **The corresponding function executes the required database query.**
3. **Results are formatted and displayed in the console.**
4. **Users can exit the application or continue exploring other features.**

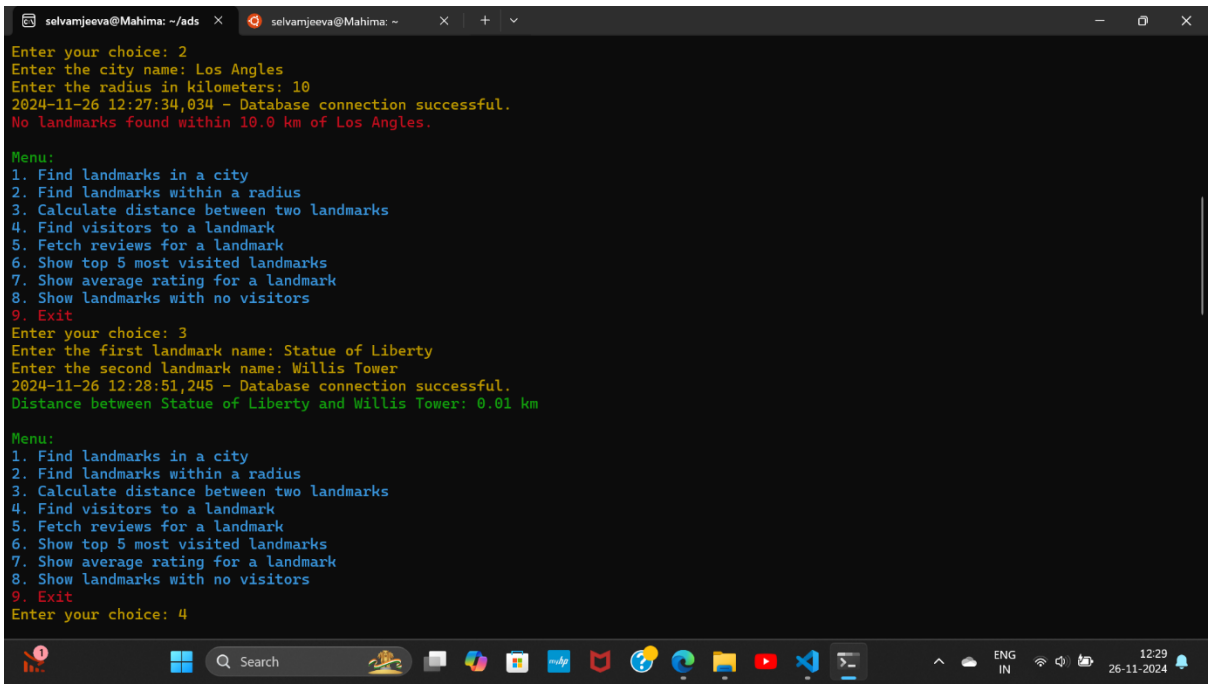
Testing

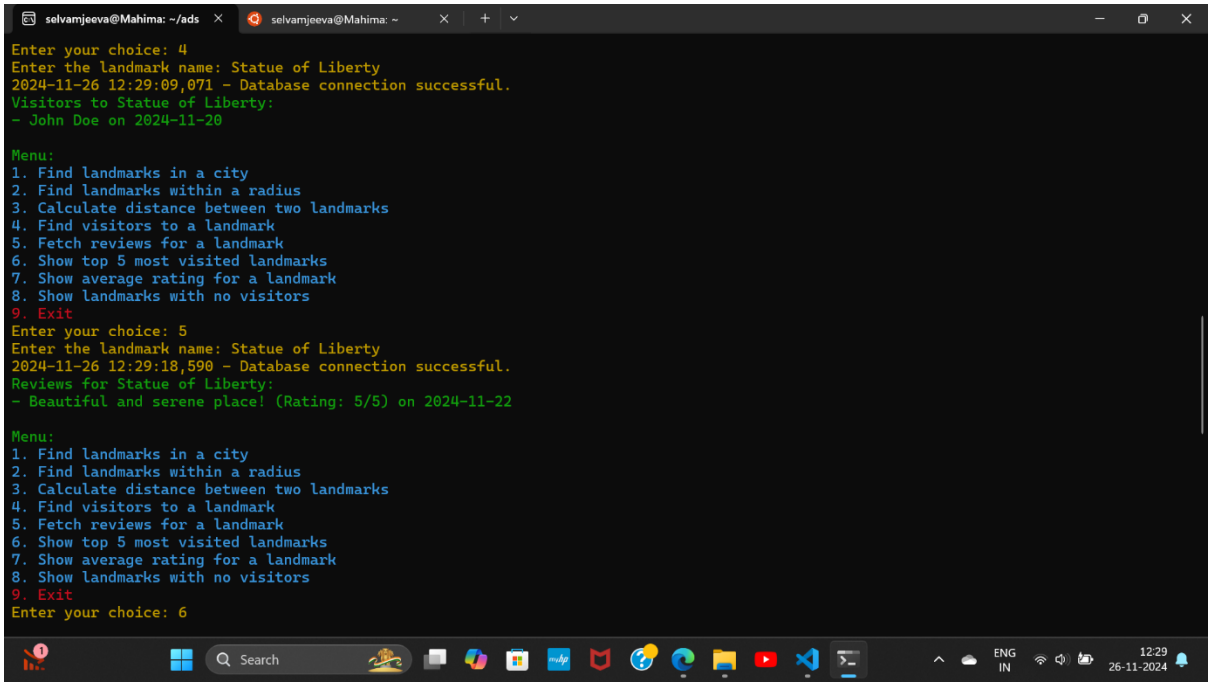
The application was tested with a sample spatial database containing:

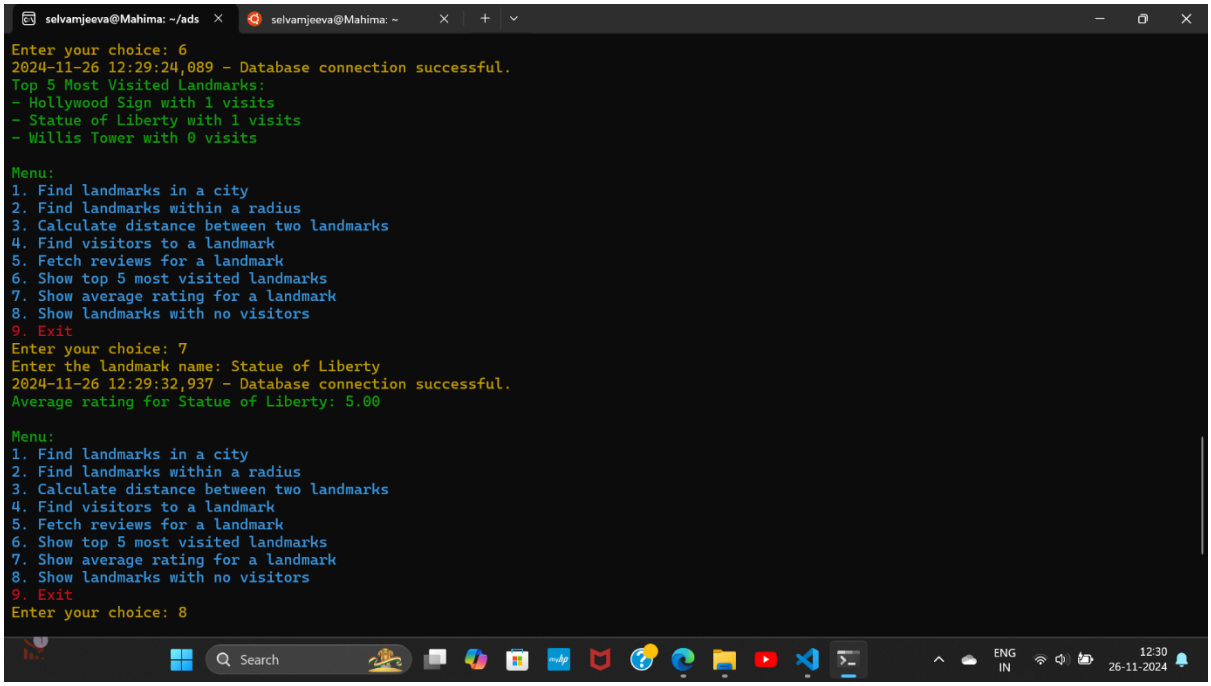
- **Cities with geographical coordinates**
- **Landmarks linked to cities**
- **Visitors and reviews associated with landmarks**

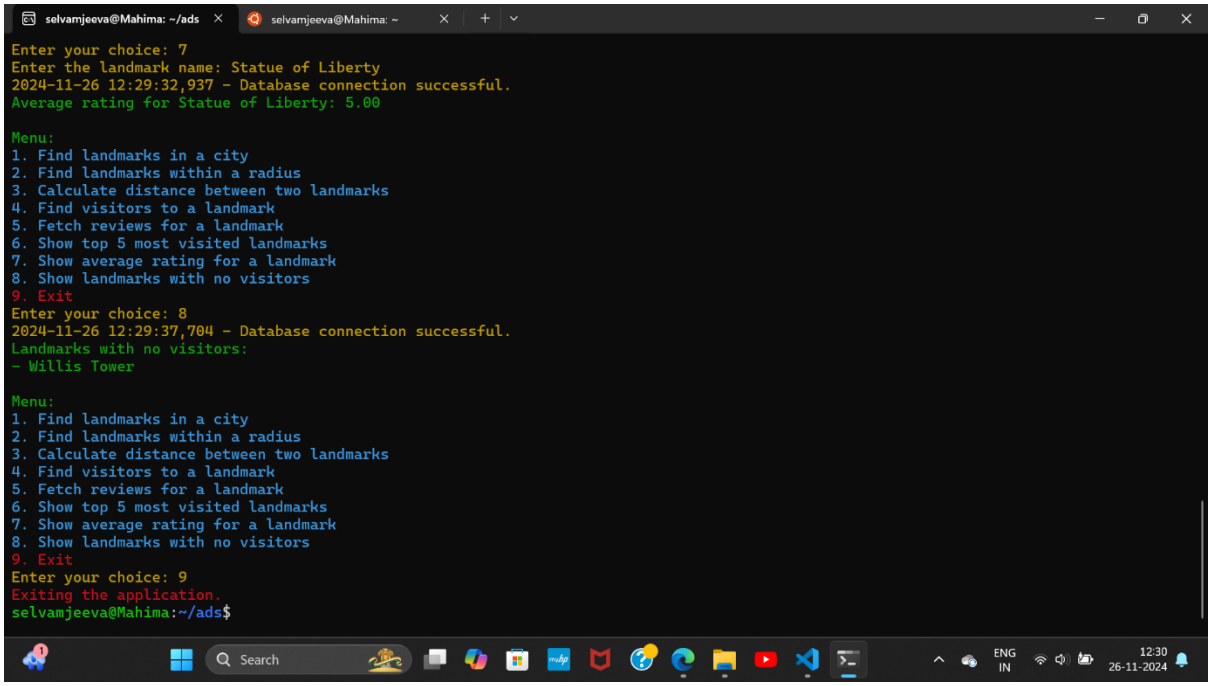
Output:


selvamjeeva@Mahima: ~/ads
selvamjeeva@Mahima: ~
selvamjeeva@Mahima:~$ cd ads
selvamjeeva@Mahima:~/ads$ python3 ads_project.py
==========
Spatial Database Application
==========
Menu:
1. Find landmarks in a city
2. Find landmarks within a radius
3. Calculate distance between two landmarks
4. Find visitors to a landmark
5. Fetch reviews for a landmark
6. Show top 5 most visited landmarks
7. Show average rating for a landmark
8. Show landmarks with no visitors
9. Exit
Enter your choice: 1
Enter the city name: New York
2024-11-26 12:27:17,636 - Database connection successful.
Landmarks in New York:
- Statue of Liberty
Menu:
1. Find landmarks in a city
2. Find landmarks within a radius
3. Calculate distance between two landmarks
4. Find visitors to a landmark
5. Fetch reviews for a landmark
6. Show top 5 most visited landmarks
7. Show average rating for a landmark
8. Show landmarks with no visitors
9. Exit
Enter your choice: 2
Search
ENG
IN
12:29
26-11-2024


selvamjeeva@Mahima: ~/ads
selvamjeeva@Mahima: ~
Enter your choice: 2
Enter the city name: Los Angles
Enter the radius in kilometers: 10
2024-11-26 12:27:34,034 - Database connection successful.
No landmarks found within 10.0 km of Los Angles.
Menu:
1. Find landmarks in a city
2. Find landmarks within a radius
3. Calculate distance between two landmarks
4. Find visitors to a landmark
5. Fetch reviews for a landmark
6. Show top 5 most visited landmarks
7. Show average rating for a landmark
8. Show landmarks with no visitors
9. Exit
Enter your choice: 3
Enter the first landmark name: Statue of Liberty
Enter the second landmark name: Willis Tower
2024-11-26 12:28:51,245 - Database connection successful.
Distance between Statue of Liberty and Willis Tower: 0.01 km
Menu:
1. Find landmarks in a city
2. Find landmarks within a radius
3. Calculate distance between two landmarks
4. Find visitors to a landmark
5. Fetch reviews for a landmark
6. Show top 5 most visited landmarks
7. Show average rating for a landmark
8. Show landmarks with no visitors
9. Exit
Enter your choice: 4
Search
ENG
IN
12:29
26-11-2024


selvamjeeva@Mahima: ~/ads
selvamjeeva@Mahima: ~
Enter your choice: 4
Enter the landmark name: Statue of Liberty
2024-11-26 12:29:09,071 - Database connection successful.
Visitors to Statue of Liberty:
- John Doe on 2024-11-20
Menu:
1. Find landmarks in a city
2. Find landmarks within a radius
3. Calculate distance between two landmarks
4. Find visitors to a landmark
5. Fetch reviews for a landmark
6. Show top 5 most visited landmarks
7. Show average rating for a landmark
8. Show landmarks with no visitors
9. Exit
Enter your choice: 5
Enter the landmark name: Statue of Liberty
2024-11-26 12:29:18,590 - Database connection successful.
Reviews for Statue of Liberty:
- Beautiful and serene place! (Rating: 5/5) on 2024-11-22
Menu:
1. Find landmarks in a city
2. Find landmarks within a radius
3. Calculate distance between two landmarks
4. Find visitors to a landmark
5. Fetch reviews for a landmark
6. Show top 5 most visited landmarks
7. Show average rating for a landmark
8. Show landmarks with no visitors
9. Exit
Enter your choice: 6
Search
ENG
IN
12:29
26-11-2024


selvamjeeva@Mahima: ~/ads
selvamjeeva@Mahima: ~
Enter your choice: 6
2024-11-26 12:29:24,089 - Database connection successful.
Top 5 Most Visited Landmarks:
- Hollywood Sign with 1 visits
- Statue of Liberty with 1 visits
- Willis Tower with 0 visits
Menu:
1. Find landmarks in a city
2. Find landmarks within a radius
3. Calculate distance between two landmarks
4. Find visitors to a landmark
5. Fetch reviews for a landmark
6. Show top 5 most visited landmarks
7. Show average rating for a landmark
8. Show landmarks with no visitors
9. Exit
Enter your choice: 7
Enter the landmark name: Statue of Liberty
2024-11-26 12:29:32,937 - Database connection successful.
Average rating for Statue of Liberty: 5.00
Menu:
1. Find landmarks in a city
2. Find landmarks within a radius
3. Calculate distance between two landmarks
4. Find visitors to a landmark
5. Fetch reviews for a landmark
6. Show top 5 most visited landmarks
7. Show average rating for a landmark
8. Show landmarks with no visitors
9. Exit
Enter your choice: 8
Search
ENG
IN
12:30
26-11-2024



Tables Used:

spatialproject=# \dt

List of relations

Schema | Name | Type | Owner

--------+----------------+-------+----------

public | cities | table | postgres

public | countries | table | postgres

public | landmark_types | table | postgres

public | landmarks | table | postgres

public | reviews | table | postgres

public | spatial_ref_sys | table | postgres

public | states | table | postgres

public | visitors | table | postgres

(8 rows)

CODE:

```
import psycopg2
import logging
from colorama import init, Fore, Back, Style
import time

init(autoreset=True)

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s')

def connect_to_db():
    try:
        connection = psycopg2.connect(
            dbname="spatialproject",
            user="mahi",
            password="mahi",
            host="localhost"
        )
        logging.info("Database connection successful.")
        return connection
    except Exception as e:
        logging.error(f"Error connecting to database: {e}")
        raise

def show_progress(message):
```

```
    print(f"{Fore.CYAN}{message}...", end="", flush=True)
    for _ in range(3):
        time.sleep(1)
        print(".", end="", flush=True)
    print(" Done!")

def display_title():
    print(Fore.GREEN + Style.BRIGHT + "============================")
    print(Fore.YELLOW + Style.BRIGHT + "Spatial Database Application")
    print(Fore.GREEN + Style.BRIGHT + "============================")

def find_landmarks_in_city(city_name):
    try:
        conn = connect_to_db()
        cursor = conn.cursor()
        query = """
            SELECT l.name
            FROM landmarks l
            JOIN cities c ON l.city_id = c.id
            WHERE c.name = %s;
        """
        cursor.execute(query, (city_name,))
        landmarks = cursor.fetchall()
        conn.close()
        return landmarks
    except Exception as e:
```

```
        logging.error(f"Error in find_landmarks_in_city: {e}")
        return []

# Query to find landmarks within a radius
def find_landmarks_within_radius(city_name, radius_km):
    try:
        conn = connect_to_db()
        cursor = conn.cursor()
        query = """
            SELECT l.name
            FROM landmarks l
            JOIN cities c ON l.city_id = c.id
            WHERE c.name = %s AND ST_DWithin(c.location, l.location, %s);
        """
        cursor.execute(query, (city_name, radius_km * 1000))
        landmarks = cursor.fetchall()
        conn.close()
        return landmarks
    except Exception as e:
        logging.error(f"Error in find_landmarks_within_radius: {e}")
        return []

def calculate_distance(landmark1, landmark2):
    try:
        conn = connect_to_db()
        cursor = conn.cursor()
```

```
        query = """
            SELECT ST_Distance(
                (SELECT location FROM landmarks WHERE name = %s),
                (SELECT location FROM landmarks WHERE name = %s)
            ) AS distance_in_meters;
        """
        cursor.execute(query, (landmark1, landmark2))
        result = cursor.fetchone()

        if result is None or result[0] is None:
            conn.close()
            return None

        distance = result[0]
        conn.close()
        return distance
    except Exception as e:
        logging.error(f"Error in calculate_distance: {e}")
        return None

def find_visitors(landmark_name):
    try:
        conn = connect_to_db()
        cursor = conn.cursor()
        query = """
            SELECT v.name, v.visit_date
```

```
            FROM visitors v
            JOIN landmarks l ON v.landmark_id = l.id
            WHERE l.name = %s;
        """
        cursor.execute(query, (landmark_name,))
        visitors = cursor.fetchall()
        conn.close()
        return visitors
    except Exception as e:
        logging.error(f"Error in find_visitors: {e}")
        return []

def fetch_reviews(landmark_name):
    try:
        conn = connect_to_db()
        cursor = conn.cursor()
        query = """
            SELECT r.review_text, r.rating, r.review_date
            FROM reviews r
            JOIN landmarks l ON r.landmark_id = l.id
            WHERE l.name = %s;
        """
        cursor.execute(query, (landmark_name,))
        reviews = cursor.fetchall()
        conn.close()
        return reviews
```

```
    except Exception as e:
        logging.error(f"Error in fetch_reviews: {e}")
        return []

def top_visited_landmarks():
    try:
        conn = connect_to_db()
        cursor = conn.cursor()
        query = """
            SELECT l.name, COUNT(v.id) AS visit_count
            FROM landmarks l
            LEFT JOIN visitors v ON l.id = v.landmark_id
            GROUP BY l.id
            ORDER BY visit_count DESC
            LIMIT 5;
        """
        cursor.execute(query)
        landmarks = cursor.fetchall()
        conn.close()
        return landmarks
    except Exception as e:
        logging.error(f"Error in top_visited_landmarks: {e}")
        return []

def average_rating(landmark_name):
    try:
```

```
        conn = connect_to_db()
        cursor = conn.cursor()
        query = """
            SELECT AVG(r.rating) AS average_rating
            FROM reviews r
            JOIN landmarks l ON r.landmark_id = l.id
            WHERE l.name = %s;
        """
        cursor.execute(query, (landmark_name,))
        avg_rating = cursor.fetchone()[0]
        conn.close()
        return avg_rating
    except Exception as e:
        logging.error(f"Error in average_rating: {e}")
        return None

def landmarks_no_visitors():
    try:
        conn = connect_to_db()
        cursor = conn.cursor()
        query = """
            SELECT l.name
            FROM landmarks l
            LEFT JOIN visitors v ON l.id = v.landmark_id
            WHERE v.id IS NULL;
        """
```

```
        cursor.execute(query)
        landmarks = cursor.fetchall()
        conn.close()
        return landmarks
    except Exception as e:
        logging.error(f"Error in landmarks_no_visitors: {e}")
        return []

def main():
    display_title()

    while True:
        print(Fore.GREEN + "\nMenu:")
        print(Fore.CYAN + "1. Find landmarks in a city")
        print(Fore.CYAN + "2. Find landmarks within a radius")
        print(Fore.CYAN + "3. Calculate distance between two landmarks")
        print(Fore.CYAN + "4. Find visitors to a landmark")
        print(Fore.CYAN + "5. Fetch reviews for a landmark")
        print(Fore.CYAN + "6. Show top 5 most visited landmarks")
        print(Fore.CYAN + "7. Show average rating for a landmark")
        print(Fore.CYAN + "8. Show landmarks with no visitors")
        print(Fore.RED + "9. Exit")

        choice = input(Fore.YELLOW + "Enter your choice: ")

        if choice == "1":
```

```
            city_name = input(Fore.YELLOW + "Enter the city name: ")
            landmarks = find_landmarks_in_city(city_name)
            if not landmarks:
                print(Fore.RED + f"No landmarks found in {city_name}.")
            else:
                print(Fore.GREEN + f"Landmarks in {city_name}:")
                for landmark in landmarks:
                    print(Fore.GREEN + f"- {landmark[0]}")
        elif choice == "2":
            city_name = input(Fore.YELLOW + "Enter the city name: ")
            radius_km = float(input(Fore.YELLOW + "Enter the radius in kilometers:
"))
            landmarks = find_landmarks_within_radius(city_name, radius_km)
            if not landmarks:
                print(Fore.RED + f"No landmarks found within {radius_km} km of
{city_name}.")
            else:
                print(Fore.GREEN + f"Landmarks within {radius_km} km of
{city_name}:")
                for landmark in landmarks:
                    print(Fore.GREEN + f"- {landmark[0]}")
        elif choice == "3":
            landmark1 = input(Fore.YELLOW + "Enter the first landmark name: ")
            landmark2 = input(Fore.YELLOW + "Enter the second landmark name:
")
            distance = calculate_distance(landmark1, landmark2)
            if distance is None:
```

```
            print(Fore.RED + f"Could not calculate the distance between {landmark1} and {landmark2}.")
        else:
            print(Fore.GREEN + f"Distance between {landmark1} and {landmark2}: {distance/1000:.2f} km")
    elif choice == "4":
        landmark_name = input(Fore.YELLOW + "Enter the landmark name: ")
        visitors = find_visitors(landmark_name)
        if not visitors:
            print(Fore.RED + f"No visitors found for {landmark_name}.")
        else:
            print(Fore.GREEN + f"Visitors to {landmark_name}:")
            for visitor in visitors:
                print(Fore.GREEN + f"- {visitor[0]} on {visitor[1]}")
    elif choice == "5":
        landmark_name = input(Fore.YELLOW + "Enter the landmark name: ")
        reviews = fetch_reviews(landmark_name)
        if not reviews:
            print(Fore.RED + f"No reviews found for {landmark_name}.")
        else:
            print(Fore.GREEN + f"Reviews for {landmark_name}:")
            for review in reviews:
                print(Fore.GREEN + f"- {review[0]} (Rating: {review[1]}/5) on {review[2]}")
    elif choice == "6":
        landmarks = top_visited_landmarks()
        if not landmarks:
```

```
            print(Fore.RED + "No landmarks found.")
        else:
            print(Fore.GREEN + "Top 5 Most Visited Landmarks:")
            for landmark in landmarks:
                print(Fore.GREEN + f"- {landmark[0]} with {landmark[1]} visits")
    elif choice == "7":
        landmark_name = input(Fore.YELLOW + "Enter the landmark name: ")
        avg_rating = average_rating(landmark_name)
        if avg_rating is None:
            print(Fore.RED + f"No reviews found for {landmark_name}.")
        else:
            print(Fore.GREEN + f"Average rating for {landmark_name}:
{avg_rating:.2f}")
    elif choice == "8":
        landmarks = landmarks_no_visitors()
        if not landmarks:
            print(Fore.RED + "All landmarks have visitors.")
        else:
            print(Fore.GREEN + "Landmarks with no visitors:")
            for landmark in landmarks:
                print(Fore.GREEN + f"- {landmark[0]}")
    elif choice == "9":
        print(Fore.RED + "Exiting the application.")
        break
    else:
        print(Fore.RED + "Invalid choice. Please try again.")
```

```
if __name__ == "__main__":
    main()
```

Conclusion:

The Spatial Database Application demonstrates the integration of spatial databases with Python for querying and analyzing geographical data. By leveraging PostgreSQL and PostGIS, it provides robust support for spatial queries and showcases the practical applications of spatial databases in real-world scenarios.