queries

December 6, 2023

1 Analysis of Boston Crimes Dataset

In this Jupyter notebook, we analyze a dataset of crimes in Boston. We'll connect to a MySQL database, execute various queries, and extract meaningful insights about crime patterns in different areas of the city. The queries are designed to provide a comprehensive view of crime trends, enabling us to understand the distribution and changes in crime rates over time.

```
[2]: # Importing the necessary libraries

import mysql.connector
import random
from pyfiglet import Figlet
```

1.1 Database Connection

We use `mysql.connector` to establish a connection with our MySQL database.

```
[3]: figlet = Figlet(font="standard")
ascii_art = figlet.renderText("BOSTON CRIMES")
print(ascii_art)
```

```
 ____   ___  ____ _____ ___  _   _    ____ ____  ___ __  __ _____ ____  
| __ ) / _ \/ ___|_   _/ _ \| \ | |  / ___|  _ \|_ _|  \/  | ____/ ___| 
|  _ \| | | \___ \ | || | | |  \| | | |   | |_) || || |\/| |  _| \___ \ 
| |_) | |_| |___) || || |_| | |\  | | |___|  _ < | || |  | | |___ ___) |
|____/ \___/|____/ |_| \___/|_| \_|  \____|_| \_\___|_|  |_|_____|____/ 
```

1.2 Query Execution Function

1.2.1 Exploring Our Python Script for Crime Data Analysis

In our group project, we've developed a Python script that features a function called `execute_query`. This function establishes a connection to a MySQL database and is capable of executing a range of queries related to crime data in Boston. The script offers an interactive experience, allowing users to select and run specific queries of interest.

1.2.2 Database Connection and Cursor Creation

- **Database Connection:** The function begins by establishing a connection to the MySQL database using `mysql.connector.connect(**db_config)`.
- **Cursor Initialization:** A cursor (`cursor = db.cursor(buffered=True)`) is created for the database connection. This cursor is used to execute queries and fetch results. The `buffered=True` parameter ensures that the server buffers the full result set and sends it to the client.

1.2.3 Handling Different Queries Based on User Choice

- **Query 1 - High Crime Areas by Time:** The user is prompted to choose between 'day' or 'night'. This choice determines the time condition in the query (`time_condition`).

  This is accomplished by joining the `Incidents` and `Place` tables and grouping the results by district and street.
- **Query 2 - Yearly Crime Trends:** This query analyzes crime trends year-on-year by calculating the percentage change in total crimes.

  The `LAG` window function is used to compare the total crimes of each year with the previous year.

  Results include the year, total crimes for that year, and the percentage change from the previous year.
- **Query 3 - Most Occurring Crime Types:** It identifies the top three most common crime types by joining the `Crime_Types` and `Incidents` tables.

  The query groups the results by crime type and orders them by the total number of occurrences, returning the top three most frequent crime types.
- **Query 4 - Area and Shootings for a Specific Street:** This query allows the user to input a street name (or choose a random one) to analyze shootings and general safety.

  It checks if the street exists in the database, and then retrieves data on shootings for that street by joining `Place` and `Incidents` and grouping by area.
- **Query 5 - Random Fact about Crime Data:** A random fact is chosen from a set of predefined queries about crime data.

  These queries provide diverse insights, such as the total number of incidents, most recent incidents, district or street with the most incidents, total number of shootings, etc.

  The script selects and executes one of these queries randomly.

1.2.4 Error Handling and Resource Management

- **Error Handling:**
  - The `try-except` block captures any errors that occur during database operations, such as connection issues or query execution errors. If an error occurs, it's printed to the console.
- **Closing Resources:**

– The `finally` block ensures that the cursor and database connection are closed properly, which is crucial for resource management and preventing memory leaks.

```
[ ]: def execute_query(db_config, query_number):
         try:
             db = mysql.connector.connect(**db_config)
             cursor = db.cursor(buffered=True)

             if query_number == 1:
                 # Query 1: Area with the most crimes by day or night
                 choice = input("Do you want to know the worst area by day or by␣
     ↪night? (day/night): ").lower()
                 if choice not in ['day', 'night']:
                     print("Invalid choice. Please enter 'day' or 'night'.")
                     return

                 time_condition = "HOUR(Incidents.Date) BETWEEN 7 AND 19" if choice␣
     ↪== 'day' else "HOUR(Incidents.Date) NOT BETWEEN 7 AND 19"
                 query = f"""
                     SELECT Place.District, Place.Street, COUNT(*) as CrimeCount
                     FROM Incidents
                     JOIN Place ON Incidents.ID = Place.Incident_ID
                     WHERE {time_condition}
                     GROUP BY Place.District, Place.Street
                     ORDER BY CrimeCount DESC
                     LIMIT 1;
                 """
                 cursor.execute(query)
                 result = cursor.fetchone()
                 if result:
                     print(f"Most crimes in {choice} time: District: {result[0]},␣
     ↪Street: {result[1]}, Number of Crimes: {result[2]}, BE CAREFUL OF THESE␣
     ↪PLACES!!")
                 else:
                     print(f"No crime data available for {choice} time.")

             elif query_number == 2:
                 # Query 2: Crime increase or decrease year on year
                 query = """
                     SELECT Year, TotalCrimes,
                            (TotalCrimes - LAG(TotalCrimes) OVER (ORDER BY Year)) /␣
     ↪LAG(TotalCrimes) OVER (ORDER BY Year) * 100 as PercentageChange
                     FROM (
                         SELECT YEAR(Date) as Year, COUNT(*) as TotalCrimes
                         FROM Incidents
                         GROUP BY YEAR(Date)
                     ) as YearlyCrimeData;
```

```
            """

            cursor.execute(query)
            results = cursor.fetchall()
            for result in results:
                year, total_crimes, percentage_change = result
                if percentage_change is not None:
                    change_description = "Increased" if percentage_change > 0␣
↪else "Decreased"
                    print(f"Year: {year}, Total Crimes: {total_crimes}, Change:␣
↪{change_description} by {percentage_change:.2f}%")
                else:
                    print(f"Year: {year}, Total Crimes: {total_crimes}, Change:␣
↪No previous year data")


        elif query_number == 3:
           # Query 3: Top 3 most occurring crime types
            query = """
                SELECT Crime_Types.Code, Crime_Types.Description,␣
↪COUNT(Incidents.ID) as TotalOccurrences
                FROM Crime_Types
                JOIN Incidents ON Crime_Types.Code = Incidents.Offence_Code
                GROUP BY Crime_Types.Code, Crime_Types.Description
                ORDER BY TotalOccurrences DESC
                LIMIT 3;
            """

            cursor.execute(query)
            results = cursor.fetchall()
            print("Top 3 most occurring crime types in the last 4 years:")
            for result in results:
                code, description, count = result
                print(f"Crime type: {description}, Occurences: {count}")

        elif query_number == 4:
            # Query 4: Area and shootings for a given street
            street_name = input("Enter a street name in CAPS (leave blank for␣
↪random examples): ").strip()

            if not street_name:
                cursor.execute("SELECT DISTINCT Street FROM Place ORDER BY␣
↪RAND() LIMIT 3;")
                random_streets = cursor.fetchall()
                print("Random street examples:")
                for street in random_streets:
                  print(street[0])
              street_name = input("Enter one of the above street names or any␣
↪other: ").strip()

          # Check if the street exists in the database
          cursor.execute("SELECT COUNT(*) FROM Place WHERE Street = %s;",␣
↪(street_name,))
          if cursor.fetchone()[0] == 0:
              print("Street not in database.")
              return

          # Fetch area and shootings for the given street
          query = """
              SELECT Place.Area, COUNT(Incidents.ID) as Shootings
              FROM Place
              LEFT JOIN Incidents ON Place.Incident_ID = Incidents.ID AND␣
↪Incidents.Shootings = 'Y'
              WHERE Place.Street = %s
              GROUP BY Place.Area;
          """

          cursor.execute(query, (street_name,))
          result = cursor.fetchone()
          if result:
              area, shootings = result
              if shootings > 0:
                  print(f"Street: {street_name}, Area: {area}, Number of␣
↪Shootings: {shootings}, Careful when going here!")
              else:
                  print("Fortunately there have been no shootings in this␣
↪area, you are safe to go!")
          else:
              print(f"Street found, but no area data available for:␣
↪{street_name}")

      elif query_number == 5:
         # Query 5: Random fact about the crime data
          queries = [
              "SELECT COUNT(*) FROM Incidents;",  # Total number of incidents␣
↪v
              "SELECT MAX(Date) FROM Incidents;",  # Most recent incident v
              "SELECT District, COUNT(*) FROM Place GROUP BY District ORDER␣
↪BY COUNT(*) DESC LIMIT 1;",  # District with the most incidents v
              "SELECT Street, COUNT(*) FROM Place GROUP BY Street ORDER BY␣
↪COUNT(*) DESC LIMIT 1;",  # Street with the most incidents v
```

```
                "SELECT COUNT(*) FROM Incidents WHERE Shootings = 'Y';",  #␣
↪Total number of shootings v
                "SELECT YEAR(Date), COUNT(*) FROM Incidents GROUP BY YEAR(Date)␣
↪ORDER BY COUNT(*) DESC LIMIT 1;",  # Year with the highest number of crimes v
                "SELECT AVG(DailyCrimes) FROM (SELECT DATE(Date) as CrimeDate,␣
↪COUNT(*) as DailyCrimes FROM Incidents GROUP BY DATE(Date)) as␣
↪DailyCrimeStats;",  # Average number of crimes per day v
                "SELECT MONTH(Date) as CrimeMonth, COUNT(*) FROM Incidents␣
↪GROUP BY MONTH(Date) ORDER BY COUNT(*) DESC LIMIT 1;"  # Month with the most␣
↪crimes v
            ]

            # Select a random query
            random_query = random.choice(queries)
            cursor.execute(random_query)
            result = cursor.fetchone()

            if result:
                if random_query == queries[0]:
                    print(f"Total number of incidents: {result[0]}")
                elif random_query == queries[1]:
                    print(f"Most recent incident date: {result[0]}")
                elif random_query == queries[2]:
                    print(f"District with the most incidents: {result[0]},␣
↪Number of Incidents: {result[1]}")
                elif random_query == queries[3]:
                    print(f"Street with the most incidents: {result[0]}, Number␣
↪of Incidents: {result[1]}")
                elif random_query == queries[4]:
                    print(f"Total number of shootings: {result[0]}")
                elif random_query == queries[5]:
                    print(f"Year with the highest number of crimes:␣
↪{result[0]}, Number of Crimes: {result[1]}")
                elif random_query == queries[6]:
                    print(f"Average number of crimes per day: {result[0]:.2f}")
                elif random_query == queries[7]:
                    print(f"Month with the most crimes: {result[0]}, Number of␣
↪Crimes: {result[1]}")

    except mysql.connector.Error as err:
        print(f"Error: {err}")
    finally:
        # Close the cursor and connection safely
        if cursor:
            cursor.close()
        if db and db.is_connected():
            db.close()
```

1.2.5 Interactive User Input

- The function is designed to interact with users, offering them a choice of queries to execute and handling their inputs to provide relevant crime data insights. This interactive nature makes the script user-friendly and accessible for non-technical users.

Starting the function, the user will have the possibility to choose from a set of 5 different queries (or to quit the function). Every time a query ir run, it goes back to the initial menu.

```
[ ]: def main():
         # Database Configuration - Update with your actual database credentials
         db_config = {
             'host': 'localhost',
             'user': 'username',
             'passwd': 'password',
             'database': 'BostonCrimes'
         }

         while True:
             print(ascii_art)
             print("\nSelect a query to execute:")
             print("1. Which areas should I avoid at different times when visiting␣
      ↪Boston? ")
             print("2. Is Boston becoming safer as the years pass?")
             print("3. What are the most common crimes to watch out for?")
             print("4. Planning on going to a certain street? Check if it is safe␣
      ↪first!")
             print("5. I feel lucky")
             print("0. Exit")

             choice = input("Enter your choice (0-5): ")

             if choice == '0':
                 break

             if choice in {'1', '2', '3', '4', '5'}:
                 execute_query(db_config, int(choice))
             else:
                 print("Invalid choice, please choose a number between 0-5.")

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

This detailed breakdown explains the intricate workings of our group's script, highlighting how it efficiently interacts with the database to provide valuable insights into crime data in Boston.