**SQL ASSIGNMENT**

**Name : PAVAN KUMAR KURVA**

**Student ID : 22065713**

**INTRODUCTION:**

**The process of data generation is a fundamental step in populating databases with relevant and meaningful information. In this context, a structured approach is essential to ensure data integrity, efficiency, and ethical considerations. This document outlines the systematic data generation process along with the database schema and ethical considerations involved.**

**DATA GENERATION PROCESS:**

1**. IMPORT LIBRARIES:** The code imports `sqlite3` for database operations and `Faker` for generating fake data.

**2. CONNECT TO DATABASE:** It establishes a connection to an SQLite database named `shoe\_database.db`.

**3. DEFINE SCHEMA:** The code defines two tables: `Brands` and `Shoes`, specifying their fields and relationships.

**4. GENERATE RANDOM DATA:** Using `Faker`, it generates fake data for brands and shoes, including brand name, country, founding year, CEO, revenue, headquarters, shoe name, shoe type, shoe size, price, and rating.

**5. INSERT DATA:** It inserts the generated data into the respective tables, ensuring that duplicate brands are not inserted.

**6. COMMIT AND CLOSE CONNECTION:** Finally, it commits the changes to the database and closes the connection.

**DATABASE SCHEMA:**

**1. BRANDS TABLE:**

- `BrandID` (Primary Key): Unique identifier for each brand.

- `BrandName`: Name of the brand.

- `Country`: Country where the brand originated.

- `FoundingYear`: Year the brand was founded.

- `CEO`: CEO of the brand.

- `Revenue`: Annual revenue of the brand.

- `Headquarters`: Location of the brand's headquarters.

- `UNIQUE (BrandName, Country) `: Compound unique key constraint to prevent duplicate brand entries.

**2. SHOES TABLE:**

- `ShoeID` (Primary Key): Unique identifier for each shoe.

- `ShoeName`: Name of the shoe.

- `ShoeType`: Type or category of the shoe.

- `ShoeSize`: Size of the shoe.

- `Price`: Price of the shoe.

- `Rating`: Rating of the shoe.

- `BrandID` (Foreign Key): References the `BrandID` in the Brands table, establishing a relationship between brands and shoes.

A screenshot of a computer

Description automatically generated

**JUSTIFICATION FOR SEPARATE TABLES AND ETHICAL DISCUSSION:**

1. **SEPARATE TABLES:** The decision to split data into separate tables for brands and shoes follows the principles of database normalization. Each table represents a distinct entity with its attributes, reducing data redundancy and improving data integrity. This separation also enables efficient querying and management of related data.

2. **ETHICAL CONSIDERATION:** Generating fake data ensures privacy and confidentiality, as real-world data may contain sensitive information about individuals or companies. Additionally, by not using real data, there is no risk of inadvertently disclosing proprietary or personal information. However, it's important to ensure that the generated data does not resemble or mimic real data closely to avoid any potential confusion or misuse.

**EXAMPLE QUERIES:**

**1. SELECTING SHOE INFORMATION:**

SELECT ShoeName, ShoeType, ShoeSize, Price, Rating

FROM Shoes

WHERE Price > 200.0 AND Rating >= 4

ORDER BY Price DESC;

**2. JOINING TABLES TO GET BRAND INFORMATION:**

SELECT s.ShoeName, s.ShoeType, s.Price, b.BrandName, b.Country

FROM Shoes s

INNER JOIN Brands b ON s.BrandID = b.BrandID

WHERE s.Rating = 5;

**3. AGGREGATE FUNCTION USAGE:**

SELECT AVG(Price) AS AvgPrice, MAX(Rating) AS MaxRating

FROM Shoes;

**4. FILTERING BASED ON DATE:**

SELECT BrandName, CEO, Revenue

FROM Brands

WHERE FoundingYear > 2000

ORDER BY Revenue DESC;

A screenshot of a computer

Description automatically generated

These example queries demonstrate different types of SQL operations such as selection, joining, aggregation, and filtering, showcasing the versatility of the database schema and the types of analysis that can be performed on the data.

**EXECUTED CODE:**

import sqlite3

from faker import Faker

fake = Faker()

conn = sqlite3.connect('shoe\_database.db')

c = conn.cursor()

# Create Brands table

c.execute('''CREATE TABLE IF NOT EXISTS Brands (

               BrandID INTEGER PRIMARY KEY,

               BrandName TEXT NOT NULL,

               Country TEXT,

               FoundingYear INTEGER,

               CEO TEXT,

               Revenue DECIMAL(12, 2),

               Headquarters TEXT,

               UNIQUE (BrandName, Country) -- Compound Unique Key

            )''')

# Create Shoes table

c.execute('''CREATE TABLE IF NOT EXISTS Shoes (

               ShoeID INTEGER PRIMARY KEY,

               ShoeName TEXT NOT NULL,

               ShoeType TEXT,

               ShoeSize REAL,

               Price DECIMAL(10, 2),

               Rating INTEGER,

               BrandID INTEGER,

               FOREIGN KEY (BrandID) REFERENCES Brands(BrandID)

            )''')

# Generate random data for Brands and Shoes

for \_ in range(1001):

    brand\_name = fake.company()

    country = fake.country()

    founding\_year = fake.random\_int(min=1800, max=2022)

    ceo = fake.name()

    revenue = fake.random.uniform(1000000.0, 1000000000.0)

    headquarters = fake.city()

    c.execute("INSERT OR IGNORE INTO Brands (BrandName, Country, FoundingYear, CEO, Revenue, Headquarters) VALUES (?, ?, ?, ?, ?, ?)",

              (brand\_name, country, founding\_year, ceo, revenue, headquarters))

    brand\_id = c.lastrowid  # Retrieve the last inserted row id

    shoe\_name = fake.catch\_phrase()

    shoe\_type = fake.word()

    shoe\_size = fake.random.uniform(6.0, 13.0)

    price = fake.random.uniform(50.0, 300.0)

    rating = fake.random\_int(min=1, max=5)

    c.execute("INSERT INTO Shoes (ShoeName, ShoeType, ShoeSize, Price, Rating, BrandID) VALUES (?, ?, ?, ?, ?, ?)",

              (shoe\_name, shoe\_type, shoe\_size, price, rating, brand\_id))

conn.commit()

conn.close()