We would like to use our 4 remaining extra hours.

Fake Data Generation and Database Population Script

Overview

This Python script generates synthetic data using the Faker library and creates corresponding CSV files for various tables in a relational database. It then uses SQLAlchemy to create an SQLite database and imports the generated CSV data into the appropriate tables. The script also includes a section to print the first row of each table in the newly created database.

Dependencies

- pandas: A data manipulation library used for handling data in tabular form.
- Faker: A library for generating fake data such as names, addresses, and other information.
- random: A built-in Python module for generating random numbers.
- string: A built-in Python module for working with string data.
- sqlalchemy: A SQL toolkit and Object-Relational Mapping (ORM) library for Python.

Fake Data Generation Functions

- `generate_username`: Generates a random username with a length of 8 characters, consisting of letters and digits.
- `generate_random_id`: Generates a random ID with a length of 10 characters, consisting of letters and digits.
- `generate_wait_times`: Generates a random time interval representing wait times.
- `generate_category`: Generates a random category from a predefined list.
- `generate_product_category`: Generates a random product category from a predefined list.
- `generate_password`: Generates a random password with a length of 15 characters, consisting of letters, digits, and punctuation.
- `generate_fake_price`: Generates a random fake price with a specified format.

CSV File Generation

- The script generates several CSV files, each representing a table in the database:

1. Friends.csv: Represents friendship data with columns 'FriendshipID', 'Status', 'Username', 'TargetedUsername', and 'StartTime'.
2. Post.csv: Represents post data with columns 'PostID', 'Username', 'Comments', 'Description', 'PostCreatedAt', and 'PostLikes'.
3. UserProfile.csv: Represents user profile data with columns like 'Username', 'FName', 'LName', 'Email', 'Phone_number', 'Address', 'DOB', 'PermLevel', 'Password', 'Gender', and 'Biography'.

4. Product.csv: Represents product data with columns 'ProductID', 'ProductName', 'ProdCategory', 'PoductDescription', and 'Price'.
5. ProductReview.csv: Represents product review data with columns 'ProductReviewID', 'Username', 'Comments', 'PRDescription', 'PRCreatedAt', 'PicorVidID', and 'PRLikes'.
6. Venue.csv: Represents venue data with columns 'VenName', 'Ven_Phone_number', 'Location', and 'Category'.
7. VenueReview.csv: Represents venue review data with columns 'VenReviewID', 'Username', 'ReviewText', 'PRDescription', 'VRCreatedAt', 'PicorVidID', 'Likes', 'Rating', 'Ven_Name', and 'WaitTimeReported'.
8. UserTimeLine.csv: Represents user timeline data with columns 'UserTimeLineID', 'PostID', 'VenReviewID', 'ProductReviewID', and 'Username'.

Database Creation and Table Population

- The script uses SQLAlchemy to create an SQLite database named 'Bool.db' and defines tables for each CSV file. Table columns are defined based on the data generated for the corresponding CSV file.

Database Population

- The script reads each CSV file and imports its data into the corresponding table in the SQLite database. If a table already exists, it replaces the existing data.

Print First Rows of Tables

- The script connects to the SQLite database and prints the first row of each table to verify that data has been successfully imported. This is to quickly ensure the script is generating data.

Step-by-step deployment

1) Required Libraries are installed.
2) Data is read into the notebook on Kaggle Notebook.
3) Connection created with SQLite to conduct queries.
4) Queries are read using pd.read_sql() method.
5) Query is stored into a Panda dataframe.
6) Each query prints resulting dataframe.

Software:

We utilize SQL Alchemy to generate an extensive database containing 1.5 million entries across 8 different tables. SQL Alchemy is a python library that connects to Oracle and uses SQLite on the backend, allowing easy manipulation of databases. This was used to then generate and manage our data using pythonic domain language, bypassing the need to use raw SQL queries to generate this code. SQL Alchemy requires initialization of a database engine, for which we used the engine bool. The engine manages connections and communicates with the data. Finally, 8 different tables are created using this Library. Data for these tables is created using the faker and the general built-in random library, commands using these libraries generates the data for use. This code is used to create 8 csv files then imports these CSV files into the SQLite database using Pandas' to_sql method.

A dictionary of Foreign Keys is generated to specify relationship between tables. SQL Alchemy object "Table" is created for each table, and it helps set up constraints for foreign keys using the foriegnkey method. SQL Alchemy's MetaData is used to organize the schema. In all, SQLAlchemy is leveraged to manage the database schema, relationships, and interaction with the SQLite database, while Pandas assists in importing data from CSV files into the database.

Functionalities:

Before implementing any commands, we download all the datafiles generated and read them in our software. This allows us to use these files for querying. The data itself is generated using faker and the built-in random library in python. These libraries are implemented using SQL Alchemy which allows use of SQLite on the backend, allowing easy manipulation of datasets.

Queries

query combining 5 tables. This query returns the phone number of all top reviewers and best venues. Top reviewers are those whose product reviews have greater than 100 likes for products with a price greater than $50, average likes for venue reviews are greater than 100.

- The five tables combined in this query are UserProfile, ProductReview, Product, VenueReview, and Venue.
- UserProfile, ProductReview, and VenueReview are all connected via reviewer's Username.
- Venue is connect to the resulting database via VenName, which was initially also present in VenueReview
- Product is connected to the resulting database via ProductID, which was initially also present in ProductReview
- Finally, after connecting these tables, all the requirements are accounted for using the where statements.

Extracting average venue rating. This query returns the average rating of each venue alongside its name and location.

- Tables VenueReview and Venue are combined using the feature VenName, which is present in both tables.
- Likes and VenName is extracted from VenueReview, where Likes is averaged and renamed as Rating.
- Resulting query is grouped by VenName and ordered by rating.

Number of friends. This query asses the number of friends each User is connected to on the social media platform.

- UserID of the target user is extracted from the table UserProfile, while the UserID of friends is extracted from table Friend.
- Number of distinct Friend UserIDs are summed to calculate the number of friends each user has.
- Friend and UserProfile tables are joined at UserID from UserProfile, where it is equivelant to FriendID in Friend table. In Friend Table, FriendID refers to userID of the original user being considered.

Returning venue information. This query returns the name of a venue, average likes and wait time that each venue has.

- The query extracts VenName, and wait_time from table Venue, and likes from table VenueReview.
- VenureReview and Venue are joined via VenName
- Likes from VenueReview are averaged, and returned alongside Venue Name and Wait times

Friends that became friends after July 20th 2020.

- Status and starttime selected from Friends table
- Date initialized to be greater than July 20th 2020 using where statement.

Venue Data. This query turns wait times of restaurants with a rating higher than 2.

- Category is extracted from Venue and Rating, Wait times, are extracted from VenueReview.
- Venue Review and Venue are joined on venue name.
- Using where statement, Rating is specified to be above 2.
- Data is grouped by venue name, and ordered by Rating.

Ratings during a period of time. This query returns all the ven_names and Rating of all venues that were rated from July 20th 1997 to July 20th 2015.

- Venue name and Rating are selected from Venue Review.
- Using where statement, data is initialized to be between July 20th 1997 to July 20th 2015
- Query is grouped by Venue name and ordered by rating.

Venue Review of each category. This query returns the average number of likes each category of venues gets.

- Likes is extracted from VeniewReview, while category is extracted from Venue.
- Venue and VenueReview are joined at VenName
- Number of likes is averaged, returned with the category

Age of user. This query identifies the age of the user when they posted a VenueReview.

- Tables Used: UserProfile and VenueReview tables are used.
- Calculation: The query uses the `TIMESTAMPDIFF` function to compute the difference in years between the user's date of birth (`u.DOB`) and the timestamp of reviews (`r. VRCreatedAt`).
- Adjustment for Birthdate & Timestamp: It utilizes `DATE_FORMAT` to compare the month and day of birth (`u.DOB`) and review timestamp (`r. VRCreatedAt`).
- Grouping & Ordering: Results are grouped by `UserID` and `r. VRCreatedAt`.
- Output: The query returns the `UserID` along with the computed `age` at the time each review was posted.

Product Rating. This query extracts the name and rating of each product.

- PRLikes is extracted and averaged as Rating from ProductReview, ProductName from Product.
- Product and ProductReview are Joined by productID
- Query is grouped by productName and ordered by likes

Retrieving information about products. name, category, average likes (treated as a rating), and price.

- ProductReview and Product tables are used.
- productName, category and price are extracted from Product, likes from ProductReview
- product and productReview are joined at productID
- Likes are averaged and returned as Rating
- Query is ordered by rating and grouped by productName.