Documentation

ABIDHA RIZWAN C A

Introduction

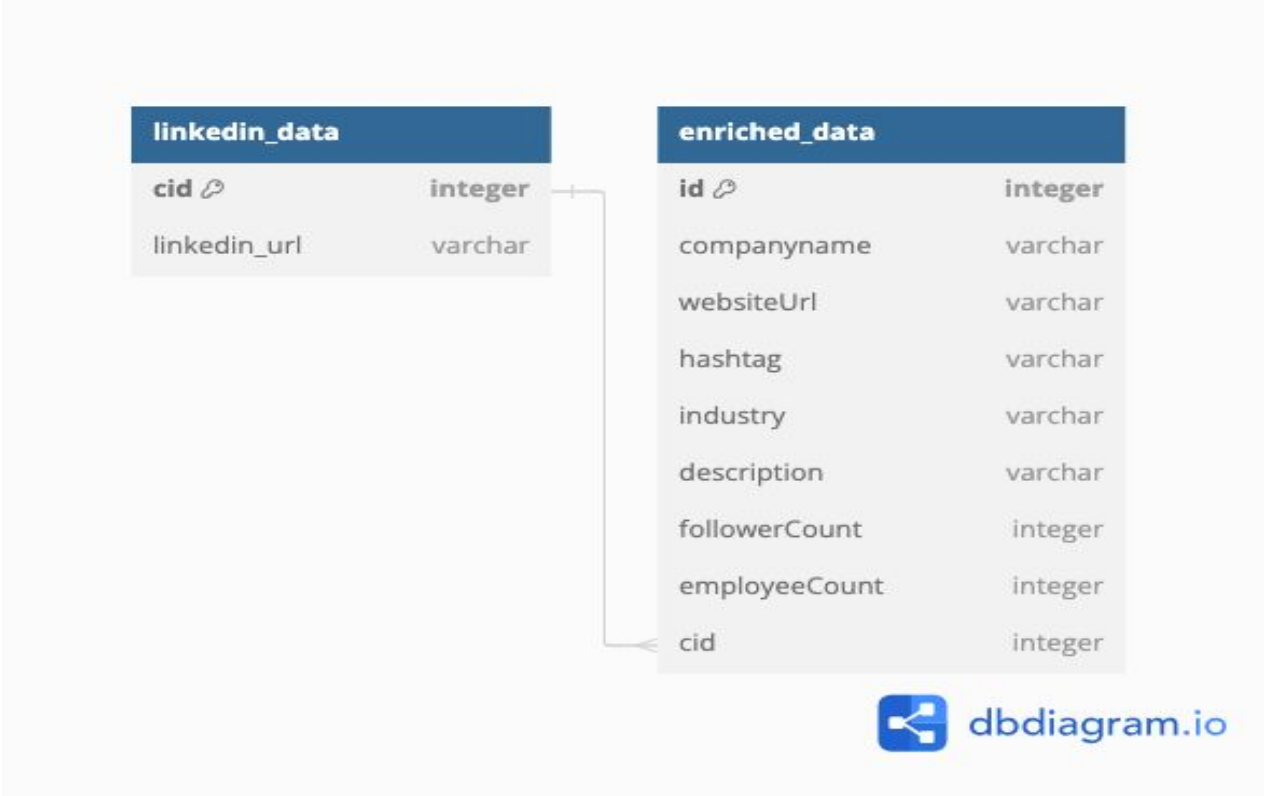
This document outlines the implementation of the Python script that supports extracts company data from an SQL database, enriches the data using the LinkedIn Bulk Data Scraper API, and stores the enriched data in a new table in the database..The script is designed to perform Extract the company_id and company_linkedin_url from the SQL database.Utilize the LinkedIn Bulk Data Scraper API, which offers a free subscription plan, to enrich the company data. Endpoint to be used rapidapi.com. such as Extract data from SQLite database, Fetch and Enrich Data using rapidapi etc.. The purpose of this document is to provide an understanding of the workflow, the structure of the database, and any assumptions or considerations made during development.

Manage and enrich company data using a SQLite database and LinkedIn API. It involves setting up a database, fetching and populating company data from LinkedIn, and providing a means to read and inspect the data. It consists of several Python scripts and SQL files:

1. **Database Setup** (`create_tables.py`)
2. **Data Fetching and Population** (`enrich.py`)
3. **SQL Table Creation** (`create.sql`)

dbdiagram

The Overall Structure depicted in a dbdiagram, by using dbdiagram.io to create this .



This dbdiagram defines two tables: linkedin_urls and enriched_data. The linkedin_urls table stores LinkedIn profile URLs with a unique identifier (id). The enriched_data table holds detailed company information, including company_name, employee_count, and various other attributes such as tagline and website_url. The foreign key relationship, linkedin_urls.id > enriched_data.company_id, links each entry in enriched_data to a corresponding LinkedIn URL from the linkedin_urls table, ensuring data integrity and associating detailed company data with its LinkedIn profile.

Steps:

1. Setup database
2. Create linkedin data table
3. Insert datas to linkedin table
4. Saves all datas
5. Create enriched data table
6. Import required libraries
7. Define constants and functions
8. Create payload for API request
9. Process and Insert Enriched Data

Work Flow

STEP1:

At the first step i created a file(company_data.py) for database setup . in this file python script initializes the SQLite by creating the necessary tables and inserting initial data. First import required library ‘sqlite3’, which provides the necessary functionality to interact with SQLite databases in Python.Then creates a function setup_database it encapsulates all the steps required to set up the database schema and initial data, The script establishes a connection to the SQLite database file named company_data.db If the file does not exist, SQLite will create it.then created a cursor object to execute SQL commands and interact with the database.

Database Schema

Step2: linkedin_data

Column Name	Data Type	Description
cid	INTEGER	Primary Key, uniquely identifies each record
linkedin_url	VARCHAR	Stores the LinkedIn URL associated with the company. This field cannot be null.

enriched_data table

Column Name	Data Type	Description
id	INTEGER	Primary key, uniquely identifies each record.
companyName	VARCHAR	Name of the company.
websiteUrl	VARCHAR	URL of the company's website.
hashtag	VARCHAR	Hashtag associated with the company.
industry	VARCHAR	Industry in which the company operates.
description	TEXT	Description of the company.
followerCount	INTEGER	Number of followers on LinkedIn.
employeeCount	INTEGER	Number of employees in the company.
cid	INTEGER	Foreign key referencing `cid` from the `linkedin_data` table, linking the two tables.

Step 3:

Insert some random data(linkedin urls) into likedin_data table in the form of tuples , Here i gave some datas for the purpose of it is a rate limited api when i calling to keep within the limit so that i put some sample datas to demonstrate this.

After that i created an another table named enriched_data. From here `linkedin_data` table will be populated with the initial set of company IDs and their LinkedIn URLs, allowing subsequent operations to reference these entries.

This setup facilitates the process of enriching company data by ensuring that each company has a unique identifier and associated LinkedIn URL for further processing or API enrichment.

Step 4:

After executing the SQL commands to create tables and insert initial data, the conn.commit() method is called to save all changes made during the current transaction to the database. This ensures that all modifications are permanently stored. Following this, conn.close() is used to close the database connection, freeing up any resources associated with it. This step concludes the setup process, ensuring that the database is properly initialized and ready for use. The if __name__ == "__main__": block ensures that the setup_database() function is executed only when the script is run directly, not when imported as a module. Then run first file using the command python3 company_data.py

Step 6:

After creating the first file i created a new file named fetch populated data(enrich.py).Then need to import some libraries os,sqlite3,requests,dotenv `.os` module, which provides a way to interact with the operating system. `sqlite3` module, enabling interaction with SQLite databases. It is used to connect to the SQLite database and execute SQL commands for retrieving and storing data.`requests` library is imported to handle HTTP requests. It simplifies the process of sending HTTP requests and receiving responses, which is essential for interacting with the LinkedIn Bulk Data Scraper API, `load_dotenv` function from the `dotenv` module. It is used to load environment variables from a `.env` file, making it possible to securely manage sensitive information such as API keys and endpoint URLs.

Step 7:

In this script, URL = os.getenv("API_ENDPOINT") and API_KEY = os.getenv("RAPID_API_KEY") fetch the API endpoint URL and authentication key from environment variables, ensuring secure and correct access to the LinkedIn Bulk Data Scraper API. The fetch_and_populate_data function is then defined to handle data retrieval and insertion processes. Within this function, a connection to the SQLite database is established using conn = sqlite3.connect("company_data.db"), and a cursor is created with cursor = conn.cursor(). This setup allows the script to execute SQL commands and manage transactions effectively.then read linkedin urls from the linkedin table .

Step 8:

Then a payload dictionary is created to include a list of LinkedIn URLs extracted from the database. This payload is used as the body of the API request to specify which URLs need to be enriched. Additionally, the headers dictionary is set up to include the API key for authentication, the host for the LinkedIn Bulk Data Scraper API, and the content type as JSON. This configuration ensures that the API request is properly formatted and authorized for data retrieval.

After creates payload dictionary sends an HTTP POST request to the API using the requests library, passing the payload and headers. The response is then parsed as JSON. If the request is successful (indicated by a status code of 200), it processes the returned data. For each company record in the response, it extracts relevant details (such as company name, ID, website, hashtag, industry, description, follower count, and employee count) and organizes them into a company_object dictionary.

Step 9:

inserts the extracted company data into the enriched_data table in the SQLite database. It uses a SQL INSERT statement to add a new record with fields such as company name, website URL, hashtag, industry, description, follower count, employee count, and company ID. The ? placeholders are replaced with values from the company_object dictionary.then saves all changes made during the current database session to the database.

Assumptions

LinkedIn Bulk Data Scraper API will remain consistently available and reliable, with properly configured API keys and endpoints. It is expected that the API will deliver data in a format that aligns with the code's requirements, allowing for accurate processing. The database schema is assumed to be correctly set up and stable, with no anticipated changes that could impact data or functionality. Additionally, it is assumed that the environment variables (API_ENDPOINT and RAPID_API_KEY) and the SQLite database (company_data.db) are correctly configured and accessible. While basic error handling is included, more comprehensive error scenarios may need to be addressed as development progresses.

Considerations

To protect data privacy and security, ensure sensitive information like API keys is not exposed in the source code and handle data according to privacy regulations. Manage API rate limits by incorporating retry logic or rate-limiting mechanisms. Optimize database performance with indexing and regular backups. Validate API data for accuracy before inserting it into the database and address any data quality issues. Design for scalability to accommodate growing data volumes and prepare for performance tuning. Implement testing and monitoring to ensure processes work correctly and to detect issues promptly.

I have pushed the data to the GitHub repository. You can view and refer to the code there:

https://github.com/abidharizwan/linkedinenrich.git

Conclusion

involves setting up and populating a database for managing company data. The `create_tables.py` script initializes the SQLite database by creating necessary tables and inserting initial data. The `enrich.py` script then enhances this data by querying the LinkedIn Bulk Data Scraper API, retrieving additional company details, and inserting them into the database. This process ensures that the database is populated with both raw and enriched company information, facilitating efficient data management and analysis.