**Project Overview**

The aim of this project was to fetch cocktail data from an online API, clean it up, and store it in a database. I wanted to make sure that the data is structured properly so that it's easy to query and use later. The key steps I used:

1. Pulling the data from the API.
2. Cleaning and organizing the data into a useful format.
3. Handling any ‘weird’ or missing data along the way.
4. Saving everything into a database (SQLite) for future use.

**1. Data Ingestion: Fetching Cocktail Data from the API**

**Why This Approach?**

We’re pulling cocktail data from the public API provided by **TheCocktailDB**. To make sure we capture everything, we query the API for cocktails starting with every letter of the alphabet (from A to Z). This way, we can get a comprehensive list of all the cocktails available.

The API URL is https://www.thecocktaildb.com/api/json/v1/1/search.php?, and we change the query to search for drinks starting with each letter (e.g., f=a for 'A' drinks, f=b for 'B' drinks, and so on).

**Why Use This Loop?**

* **Efficiency**: The API only returns a limited number of results at a time, so we loop through all the letters to make sure we don’t miss any cocktails.
* **Scalability**: The number of drinks available might vary by letter (more cocktails starting with 'A' than 'Z'), so looping through each letter ensures we cover everything without overloading the system.

**Why It Works Well**

* **Keeps Things Modular**: By querying one letter at a time, the code stays simple and easy to test.
* **Quick and Efficient**: We get all the data without needing to fetch unnecessary info.

**2. Data Transformation: Cleaning and Normalizing the Data**

**Why We Do This**

When we get the data from the API, it’s in a little messy format. We want to clean it up and store it in a way that makes sense. We’ve split the data into two tables:

* **Drinks Table**: This holds general info about each drink (name, category, instructions, etc.).
* **Ingredients Table**: This holds all the ingredients and their measurements for each drink.

**Normalization**

By separating the ingredients into their own table, we avoid having to repeat the same ingredient data every time the drink appears. This keeps the database clean and reduces redundancy.

**Why Normalize?**

* **No Redundancy**: If we put ingredients directly in the Drinks table, we’d be repeating ingredient info for each cocktail that uses the same ingredient (like “lime” or “vodka”). Normalizing prevents that.
* **Easier to Query**: If we need to find all the drinks that contain “lime,” it’s much easier to query the Ingredients table rather than pulling up every drink record and checking if it has “lime” listed.

**3. Handling Edge Cases**

**Why It's Important**

Not all cocktail data is perfect—some drinks might be missing ingredients, or the ingredients and their measurements might not match up. Our job is to make sure this doesn’t mess up our database.

* **If Ingredients Don’t Match Measures**: If there are more ingredients than measures (or vice versa), we skip that cocktail entirely. We want to make sure the data we store is valid.

**Why This Makes Sense**

* **Keeps Data Clean**: This ensures that we only store complete, valid drinks in our database.
* **Makes It More Robust**: This way, if the API gives us messy data, our script won’t break or insert garbage into the database.

**4. Storing the Data in SQLite**

**Why SQLite?**

After cleaning up the data, we store it in SQLite database. SQLite is a great choice for this kind of project because:

* **Lightweight**: It doesn’t require a server or complex setup.
* **Portable**: The database is stored as a single file, so it’s easy to share and move around.
* **Simple**: SQLite is perfect for small to medium projects like this, where we just need to store structured data without a full-blown server.

We create two tables in SQLite:

* **Drinks Table**: This stores information like the drink's name, instructions, category, etc.
* **Ingredients Table**: This stores ingredients and measurements for each drink.

**Why Use SQLite?**

* **No Overhead**: It’s easy to set up and doesn’t need a complex server setup.
* **Portable**: Everything’s in one file, so it’s easy to share or back up.
* **Perfect for This Project**: Since we don’t need a huge database system, SQLite fits the bill perfectly.

**Data Integrity**

Once the data is ready, we commit it to the database in a single transaction. This ensures everything is saved correctly. The script uses if\_exists='replace' to make sure the database is replaced each time the script runs. This is helpful when we want to update the database with fresh data. If we wanted to add new data without replacing the old, we could change it to if\_exists='append'.

**Conclusion**

This code was designed to be:

* **Simple**: It pulls data from an API and stores it in a clean, usable format.
* **Efficient**: It normalizes the data to reduce redundancy and makes it easy to query.
* **Robust**: It handles weird cases (like missing or mismatched ingredients) to make sure we only save good data.
* **Portable**: By using SQLite, the data is stored in a simple, easy-to-use format.