# COCKTAILS DB CASE STUDY

**APPROACH:**

This section discusses the approach taken to solve the case study. API calls were made to download data per letter name. There is only a handful amount of options when it comes to making calls to Cocktails API. The most efficient (in my opinion) was to make an API call using the first alphabet to get the max records with minimum calls. These responses are treated as batches, processed on the fly, and loaded to the SQLite database in each iteration. Each batch is logged with a unique batch_id. All the logs are stored in the "**cocktails_logs.txt**" file.

For the design of the SQLite cocktails database, LunaModeler was used. The ERD is also attached (**ERD_cocktails_db.png**). The database was created via DB Browser for SQLite, all the DDL SQL commands are present in "**cocktails_sql.db**". "**main.py**" is the Python file that makes the API calls to Cocktails API, processes the records using functions present in "**utils.py**" and logs and loads the batches in the aforementioned SQLite database.

After loading all the processed/available records the ad-hoc SQL queries are run to answer the asked questions in the case study. All of these queries are present in the "**queries.sql**" file.

**NOTE:** Currently the entire database is not populated, "drink_media_info", "drink_tags", "drink_iba_tags" along with their hierarchical normalized tables are kept empty because of time constraints and not required to answer questions asked in ad-hoc SQL queries.

**Environment:**

**drink-master.yml** file contains the requirements to run this project. Assuming *conda* is already installed, run the following command from the root of this folder.

**conda env create -f drink-master.yml**

**OPPORTUNITIES FOR ENHANCEMENT:**

This section highlights areas in which the project or codebase can be improved further. It serves as a reference for potential optimizations (to the best of my knowledge) that can be implemented to make this case study more efficient. Most of the suggestions are made while keeping in mind the scalability aspect of this task in mind.

1. Better data extraction:
    a. The improved API call to get paginated responses
    b. Save the returned data in the raw format first in a staging area (Data Lake) so it can be referenced in the future if needed
2. Replace redundant code (esp in **get_foriegn_key()* methods)
3. Strict data validation for data types following an agreed-upon data contract before processing the record and loading it into the database.
4. Save/download/store the unprocessed records (e.g records with no DE instructions in this case) somewhere for further investigation
5. Setup alerting and send monitor event for "ERROR" type logs
6. Type hints and annotations for better-documented code especially in functions.
7. Better optimized database design