Pokemon Data API Assignment

Overview

This assignment involves working with a SQLite database containing Pokémon data. You will need to implement functions to connect to the database, clean the data, and create a FastAPI application with several endpoints to query the cleaned data.

The main file you will be working with is `candidate_solution.py setup_db.py`.

Prerequisites

- Python 3.6+
- SQLite3 (usually included with Python)
- FastAPI: `pip install fastapi`
- Uvicorn: `pip install uvicorn[standard]`

Setup

1. Ensure you have Python and the required libraries installed.
2. Run `python setup_db.py` to create the initial database (pokemon_assessment.db) and examine its contents using a tool like DB Browser for SQLite to understand the "dirty" data you need to clean.

Tasks

You need to complete the implementation sections marked with `# --- Implement Here ---` within the `candidate_solution.py` file.

NOTE: Ensure work is ONLY placed inside of the marked sections. Alongside a human review of your code, your test will also be programmatically evaluated and work done outside of the marked sections **will not** be considered.

Task 1: Connect to the Database

- **Function:** `connect_db()`
- **Objective:** Implement the logic to establish a connection to the `pokemon_assessment.db` SQLite database.
- **Requirements:**
  - Use the `sqlite3` library.
  - The function should return the `sqlite3.Connection` object if successful.
  - The function already checks if the database file exists; you need to add the connection logic within the `try` block.
  - Return `None` if any `sqlite3.Error` occurs during connection.

Task 2: Clean the Database

- **Function:** `clean_database(conn: sqlite3.Connection)`
- **Objective:** Implement the logic to clean the data within the connected database.

- **Requirements:**
  - Use the provided `conn` (database connection) object and its cursor.
  - **Remove Duplicates:** Identify and remove duplicate entries from the `pokemon`, `types`, `abilities`, and `trainers` tables based on their names (case-insensitive). Choose a consistent strategy for keeping one record (e.g., keep the one with the lowest `id`).
  - **Correct Misspellings:** Implement `UPDATE` statements to fix known specific misspellings across all tables. You can use the following resources as reference: Pokemon Wiki, Pokemon API
  - **Standardize Casing:** Update names in `pokemon`, `types`, `abilities`, and `trainers` tables to use a consistent casing (e.g., Title Case like 'Fire', 'Ash', 'Static', 'Pikachu'). Handle potential variations in existing casing.
  - **Remove Redundant Data:** Delete specific redundant or placeholder entries if they exist (e.g., types named '---', '???', or '').
  - Ensure all changes are committed to the database upon successful completion.
  - Implement error handling using the `try...except` block and rollback changes if an error occurs.

Task 3: Basic Root Endpoint

- **Function:** `read_root()` within `create_fastapi_app()`
- **Objective:** Implement the root (`/`) endpoint for the FastAPI application.
- **Requirements:**
  - Return a simple JSON response.
  - The JSON object must contain a key named `message` with any string value.

Task 4: Get Pokémon by Ability

- **Function:** `get_pokemon_by_ability(ability_name: str)` within `create_fastapi_app()`
- **Objective:** Create an endpoint `/pokemon/ability/{ability_name}` that returns a list of Pokémon names possessing the specified ability.
- **Requirements:**
  - Query the *cleaned* database.
  - The query should join relevant tables to find Pokémon associated with the given `ability_name` (case-insensitive comparison isrecommended).
  - Return a JSON list of Pokémon names (`List[str]`).
  - Handle edge cases

Task 5: Get Pokémon by Type

- **Function:** `get_pokemon_by_type(type_name: str)` within `create_fastapi_app()`
- **Objective:** Create an endpoint `/pokemon/type/{type_name}` that returns a list of Pokémon names belonging to the specified type.
- **Requirements:**
  - Query the *cleaned* database.
  - The query should consider both `type1_id` and `type2_id` in the `pokemon` table.
  - Return a JSON list of unique Pokémon names (`List[str]`).
  - Handle edge cases

Task 6: Get Trainers by Pokémon

- **Function:** `get_trainers_by_pokemon(pokemon_name: str)` within `create_fastapi_app()`
- **Objective:** Create an endpoint `/trainers/pokemon/{pokemon_name}` that returns a list of trainer names who own the specified Pokémon.
- **Requirements:**
  - Query the *cleaned* database.
  - Join relevant tables to find trainers associated with the given `pokemon_name` (case-insensitive comparison is recommended).
  - Return a JSON list of unique trainer names (`List[str]`).
  - Handle edge cases

Task 7: Get Abilities by Pokémon

- **Function:** `get_abilities_by_pokemon(pokemon_name: str)` within `create_fastapi_app()`
- **Objective:** Create an endpoint `/abilities/pokemon/{pokemon_name}` that returns a list of ability names for the specified Pokémon.
- **Requirements:**
  - Query the *cleaned* database.
  - Join relevant tables to find abilities associated with the given `pokemon_name` (case-insensitive comparison is recommended).
  - Return a JSON list of unique ability names (`List[str]`).
  - Handle edge cases

Task 8: Create Pokémon record via API

- **Function:** `your_custom_method_name` within `create_fastapi_app()`
- **Objective:** Create an endpoint that will take a pokemon name and create a new record in the `trainer_pokemon_abilities` table with all the relevant foreign key connections. You will need to utilize the Pokemon API specifically the `https://pokeapi.co/api/v2/pokemon/` endpoint to fetch the pokemon's relevant data. The endpoint will return the new records database ID
- **Requirements:**
  - Create a custom endpoint and function that only takes the Pokémon name as an input.
  - The function will connect to and query the Pokémon API to fetch the relevant data for the given pokemon name
  - Create a new record in the `trainer_pokemon_abilities` table.
    - Create/ reference the appropriate foreign keys.
    - You may randomly assign a trainer to the record as the API does not provide Trainer details.
  - Handle edge cases

Task 9: Submission

Please upload ONLY the completed `candidate_solution.py` file to your public Github repository and provide us a link to it.

Running the Application (Locally)

The script includes a `if __name__ == "__main__":` block. You can run the script directly using:

```
python candidate_solution.py
```

This will:

- Attempt to connect to the database.
- Run the clean_database function.
- Start the FastAPI application using Uvicorn on http://127.0.0.1:8000.
- You can then access the API endpoints in your browser or using tools like curl, Postman.

Evaluation Criteria

Your solution will be evaluated based on:

- **Correctness**:
  - Does the code successfully complete all tasks as specified?
  - Do the API endpoints return the expected data based on the cleaned database?
- **Data Cleaning Logic**:
  - Is the data cleaning comprehensive and correct according to the requirements (duplicates, misspellings, casing)?
  - Are foreign keys handled correctly during deduplication?
  - Database Interaction: Are database connections and transactions handled properly (including commits and rollbacks)?
  - Are queries efficient?
- **API Implementation**:
  - Are the FastAPI endpoints implemented correctly, handling path parameters and returning the specified response models?
  - Code Quality: Is the code clean, well-formatted, readable, and maintainable?