Step 1: API Key Acquisition & Data Extraction

The first step in the NASA Near-Earth Object (NEO) Tracking Project focused on **connecting to NASA's public Asteroids NeoWs API** to gather data.

🔑NASA API Key Acquisition

- Navigated to https://api.nasa.gov and filled out the registration form.
- Received a unique API key via email.
- Constructed the request URL using the following format:

```
https://api.nasa.gov/neo/rest/v1/feed?start_date=YYYY-MM-DD&end_date=YYYY-MM-DD&api_key=YOUR_KEY
```

🕐Targeted Data Extraction:

- **Start Date:** 2024-01-01
- **End Date:** 2024-01-07 (7-day range)
- Used Python's `requests` library to make API calls.

🔦Pagination Handling:

- Implemented dynamic pagination using the `data['links']['next']` field in the API response.
- Continued fetching 7-day segments until **10,000 asteroid records** were collected.
- All data was stored in a list of dictionaries for further processing.

🔗Outcome:

- Successfully extracted raw JSON data containing asteroid details and their close approach events.

Step 2: Field Extraction & Data Cleaning

The second step ensured the raw JSON data was cleaned and transformed for SQL insertion.

🗒️Fields Extracted from JSON:

1. `id`
2. `neo_reference_id`
3. `name`
4. `absolute_magnitude_h`
5. `estimated_diameter_min_km`
6. `estimated_diameter_max_km`
7. `is_potentially_hazardous_asteroid`

8. `close_approach_date`
9. `relative_velocity_kmph`
10. `astronomical` (miss distance in AU)
11. `miss_distance_km`
12. `miss_distance_lunar`
13. `orbiting_body`

🧹Data Cleaning Process:

- **Data Type Conversion:** Ensured fields were cast to correct types (`float`, `int`, `boolean`, `datetime`).
- **Date Standardization:** Used Python's `datetime.strptime()` to convert string dates into `datetime.date` objects.
- **Missing Values Handling:** Used `.get()` method to handle missing keys gracefully. Defaults applied or records skipped.
- **Field Renaming:** Cleaned up ambiguous names like `astronomical(AU)` → `astronomical_au`.

📦Prepared Data Structure:

- Created structured Python dictionaries ready for SQL insertion.
- Final data organized as a list: `asteroids_data = [dict1, dict2, ..., dictN]`

🔗Outcome:

Cleaned and formatted data that aligns with the schema of two SQL tables (`asteroids` and `close_approach`) for seamless insertion in Step 3.