## Step 3: SQL Insertion & Table Creation

The third phase of the NASA Near-Earth Object (NEO) tracking project involved setting up a **relational database structure** to organize and store the cleaned data. This step ensures efficient querying, analysis, and dashboard development.

### 📂Database Setup:

- **Database Name:** `asteroids_data_db`
- **Tool Used:** SQLite (via Python's `sqlite3` module)

---

### 🔍Outline: Table Creation

Two tables were created based on the cleaned JSON fields:

#### 🗜️Table 1: `asteroids`

Stores general information about each asteroid.

```
CREATE TABLE IF NOT EXISTS asteroids (
    id INT,                         -- Not unique; duplicates exist across
events
    name TEXT NOT NULL,
    absolute_magnitude_h REAL,
    estimated_diameter_min REAL,
    estimated_diameter_max REAL,
    is_potentially_hazardous_asteroid BOOLEAN,
    nasa_jpl_url TEXT,
    sentry_object BOOLEAN
);
```

#### ⏱️Table 2: `close_approach`

Stores event-based details about each asteroid's approach.

```
CREATE TABLE IF NOT EXISTS close_approach (
    approach_id INTEGER PRIMARY KEY,
-- Unique ID for each approach event
    neo_reference_id INTEGER,                  -- References asteroid ID
    close_approach_date TEXT,                  -- Stored in YYYY-MM-DD format
    relative_velocity_kmph REAL,
    astronomical_au REAL,                      -- Renamed for compatibility
    miss_distance_km REAL,
    miss_distance_lunar REAL,
    orbiting_body TEXT,
    FOREIGN KEY (neo_reference_id) REFERENCES asteroids(id)
);
```

---

## 📈Data Verification:

After creation, each table was verified using SQL queries:

```
cursor.execute("SELECT * FROM asteroids")
data = cursor.fetchall()
columns = [i[0] for i in cursor.description]
new_df = pd.DataFrame(data, columns=columns)
new_df  # Displays 6+ columns of asteroid data

cursor.execute("SELECT * FROM close_approach")
data = cursor.fetchall()
columns = [i[0] for i in cursor.description]
new_df = pd.DataFrame(data, columns=columns)
new_df  # Displays 7+ columns of approach data
```

---

## 🔗Data Insertion:

### 🔧Inserting into `asteroids` table:

```
insert = "INSERT INTO asteroids VALUES (?, ?, ?, ?, ?, ?, ?, ?)"

for i in asteroids_data:
    values = (
        i['id'],
        i['name'],
        i['absolute_magnitude_h'],
        i['estimated_diameter_min_km'],
        i['estimated_diameter_max_km'],
        i['is_potentially_hazardous_asteroid'],
        i['nasa_jpl_url'],
        i['sentry_object']
    )
    cursor.execute(insert, values)

connection.commit()
```

### 🔧Inserting into `close_approach` table:

```
insert = "INSERT INTO close_approach VALUES (?, ?, ?, ?, ?, ?, ?, ?)"

for i in asteroids_data:
    values = (
        i['approach_id'],
        i['neo_reference_id'],
        i['close_approach_date'],
        i['relative_velocity_kmph'],
        i['astronomical'],
        i['miss_distance_km'],
        i['miss_distance_lunar'],
        i['orbiting_body']
    )
    cursor.execute(insert, values)

connection.commit()
```

---

## 📊Post-Insertion Validation:

To confirm the records were successfully inserted:

```
# Asteroids Table
cursor.execute("SELECT * FROM asteroids")
data = cursor.fetchall()
columns = [i[0] for i in cursor.description]
new_df = pd.DataFrame(data, columns=columns)
new_df.head()  # Confirms 6+ relevant columns

# Close Approach Table
cursor.execute("SELECT * FROM close_approach")
data = cursor.fetchall()
columns = [i[0] for i in cursor.description]
new_df = pd.DataFrame(data, columns=columns)
new_df.head()  # Confirms 7+ relevant columns
```

---

## 🔗Outcome:

The successful creation and population of the `asteroids` and `close_approach` tables marked the transition to the next phase of the project: **SQL querying and dashboard development using Streamlit**.