```python
import requests
import psycopg2
from datetime import datetime

# Configuration de la base de données PostgreSQL
DB_CONFIG = {
    'dbname': 'dbname',
    'user': 'username',
    'password': 'password',
    'host': 'host'
}

# Configuration de l'API Flightradar24
FLIGHT_LIST_URL = "https://api.flightradar24.com/common/v1/flight/list.json"
FLIGHT_PLAYBACK_URL = "https://api.flightradar24.com/common/v1/flight-
playback.json"

def get_last_10_flights(registration):
    """Récupère les 10 derniers vols pour une immatriculation donnée."""
    params = {
        'query': registration,
        'fetchBy': 'reg',
        'page': 1,
        'limit': 10
    }
    response = requests.get(FLIGHT_LIST_URL, params=params)
    response.raise_for_status()
    data = response.json()
    return data['result']['response']['data'][:10]

def get_flight_details(flight_id):
    """Récupère les détails du vol incluant la trajectoire."""
    params = {'flightId': flight_id}
    response = requests.get(FLIGHT_PLAYBACK_URL, params=params)
    response.raise_for_status()
    data = response.json()
    return data['result']['response']['data']['flight']

def detect_phases(track, flight_info):
    """Détecte les phases de vol à partir des données de trajectoire."""
    sorted_track = sorted(track, key=lambda x: x['timestamp'])
    phases = []

    # Taxi-out
    taxi_out_start = sorted_track[0]['timestamp']
    taxi_out_end = next((p['timestamp'] for p in sorted_track if p['altitude'] >
0), None)
    if not taxi_out_end:
        return []
    phases.append(('taxi-out', taxi_out_start, taxi_out_end))
```

```
    # Takeoff
    takeoff_start = taxi_out_end
    takeoff_end = None
    max_alt = 0
    for i, p in enumerate(sorted_track):
        if p['timestamp'] < takeoff_start:
            continue
        if p['altitude'] > max_alt:
            max_alt = p['altitude']
        else:
            if all(sp['altitude'] <= max_alt for sp in sorted_track[i:i+3]):
                takeoff_end = p['timestamp']
                break
    takeoff_end = takeoff_end or sorted_track[-1]['timestamp']
    phases.append(('takeoff', takeoff_start, takeoff_end))

    # Cruise
    cruise_start = takeoff_end
    cruise_end = None
    for i, p in enumerate(sorted_track):
        if p['timestamp'] < cruise_start:
            continue
        if i > 0 and p['altitude'] < sorted_track[i-1]['altitude']:
            cruise_end = sorted_track[i-1]['timestamp']
            break
    cruise_end = cruise_end or sorted_track[-1]['timestamp']
    phases.append(('cruise', cruise_start, cruise_end))

    # Landing
    landing_start = cruise_end
    landing_end = next((p['timestamp'] for p in sorted_track if p['timestamp'] >=
landing_start and p['altitude'] <= 0), sorted_track[-1]['timestamp'])
    phases.append(('landing', landing_start, landing_end))

    # Taxi-in
    taxi_in_start = landing_end
    taxi_in_end = sorted_track[-1]['timestamp']
    phases.append(('taxi-in', taxi_in_start, taxi_in_end))

    # Format des résultats
    flight_name = f"{flight_info['aircraft']['registration']} -
{flight_info['identification']['number']['default']} -
{datetime.fromtimestamp(flight_info['time']['scheduled']['departure']).strftime('%Y
-%m-%d')}"
    return [{
        'name': flight_name,
        'type': phase[0],
        'start': datetime.fromtimestamp(phase[1]),
        'end': datetime.fromtimestamp(phase[2])
    } for phase in phases]
```

```python
def save_to_db(phases):
    """Sauvegarde les phases de vol dans PostgreSQL."""
    conn = psycopg2.connect(**DB_CONFIG)
    cursor = conn.cursor()
    for phase in phases:
        cursor.execute("""
            INSERT INTO flight_phases (name, phase_type, start_time, end_time)
            VALUES (%s, %s, %s, %s)
        """, (phase['name'], phase['type'], phase['start'], phase['end']))
    conn.commit()
    cursor.close()
    conn.close()

def main(registration):
    flights = get_last_10_flights(registration)
    for flight in flights:
        flight_id = flight['identification']['id']
        try:
            flight_details = get_flight_details(flight_id)
            track = flight_details['track']
            phases = detect_phases(track, flight_details)
            if phases:
                save_to_db(phases)
        except Exception as e:
            print(f"Erreur lors du traitement du vol {flight_id}: {e}")

if __name__ == "__main__":
    import sys
    if len(sys.argv) != 2:
        print("Usage: python flight_phases.py <registration>")
        sys.exit(1)
    main(sys.argv[1])
```