Application 1: Rule Engine with AST

Objective:

To develop a simple 3-tier rule engine application that determines user eligibility based on attributes like age, department, income, and spending using Abstract Syntax Tree (AST). The system allows dynamic rule creation, modification, and combination.

Steps for Implementation:

1. **Data Structure for AST**:
   - A `Node` structure will be used with fields: `type` (operator or operand), `left` (left child), `right` (right child), and `value` (operand value).

   Example Code:

   ```
   python
   Copy code
   class Node:
       def __init__(self, node_type, left=None, right=None, value=None):
           self.type = node_type
           self.left = left
           self.right = right
           self.value = value
   ```

2. **Database**:
   - Define schema for storing rules.
   - Example database table structure for rules:

     ```
     sql
     Copy code
     CREATE TABLE Rules (
         id INT PRIMARY KEY,
         rule_name VARCHAR(255),
         ast_representation TEXT
     );
     ```

3. **API Design**:
   - **`create_rule(rule_string)`**: Converts a rule string into an AST.
   - **`combine_rules(rules)`**: Combines multiple ASTs efficiently.
   - **`evaluate_rule(json_data)`**: Evaluates the rule against user data.

   Example Code for Rule Creation:

   ```
   python
   Copy code
   def create_rule(rule_string):
       # Parse rule string into AST
       # Example parser logic here
       pass
   ```

4. **Test Cases**:
    - Create rules and validate AST representations.
    - Test combining rules and evaluating them with sample data.

Bonus Features:

- Error handling for invalid rules, modification of existing rules, and user-defined functions.

Application 2: Real-Time Data Processing System for Weather Monitoring

Objective:

Develop a real-time weather data processing system using the OpenWeatherMap API. The system fetches weather data, converts temperatures to Celsius, provides daily summaries, and triggers alerts based on user-defined thresholds.

Steps for Implementation:

1. **API Integration**:
    - Retrieve weather data every 5 minutes using OpenWeatherMap API.

    Example Code:

    ```
    python
    Copy code
    import requests

    def get_weather_data(api_key, city):
        url =
    f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_ke
    y}"
        response = requests.get(url)
        return response.json()
    ```

2. **Temperature Conversion**:
    - Convert temperatures from Kelvin to Celsius:

    ```
    python
    Copy code
    def kelvin_to_celsius(kelvin_temp):
        return kelvin_temp - 273.15
    ```

3. **Rollups and Aggregates**:
    - Daily weather summary: calculate average, max, min temperatures, and dominant condition.
    - Example database schema:

      ```
      sql
      Copy code
      CREATE TABLE DailyWeatherSummary (
          date DATE,
          avg_temp FLOAT,
          max_temp FLOAT,
          min_temp FLOAT,
          dominant_condition VARCHAR(50)
      );
      ```

4. **Alerting Mechanism**:

- Define thresholds (e.g., temp > 35°C for 2 consecutive updates) and trigger alerts.
- Example:

```
python
Copy code
def check_thresholds(temp, threshold):
    if temp > threshold:
        print("Alert: Temperature exceeded!")
```

5. **Visualization**:
   - Generate visualizations for daily summaries and historical data.

Bonus Features:

- Support for additional weather parameters, forecast retrieval, and advanced alerting.
