Aquatic Labs Take Home Project

You'll build a minimal backend system that ingests sensor data (temperature and conductivity measurements arriving twice every second from three sensors), stores the raw measurements, and computes rolling statistical aggregations for efficient query responses.

Your system should provide a RESTful API with endpoints for querying raw measurements by sensor and time range, and accessing aggregated statistics of the measurements.

Your system should compute and store aggregated statistics at multiple time resolutions—one-minute windows for the most recent hour and five-minute windows for anything past that—automatically downsampling as measurements age to balance storage efficiency with query performance. Your API should select the appropriate resolution based on the requested time range, handling the trade-off between resolution and data volume.

This is a backend-only project with no frontend requirements—deliver working endpoints, an implemented database schema, clear documentation of your architectural decisions, and a README with setup instructions that would allow someone to test your API using curl or Postman.

Below is a simulated sensor script, which will send mock measurements to port 5000 on localhost.

```
"""
sensor_simulator.py

Simulates 3 water quality sensors sending temperature and conductivity measurements.
Sends data to http://localhost:5000/measurements every 5 seconds.

Usage:
   python sensor_simulator.py

Requirements:
   pip install requests
"""

import requests
import time
import random
from datetime import datetime

# Configuration
```

```
API_URL = 'http://localhost:5000/measurements'
SENSORS = ['sensor_001', 'sensor_002', 'sensor_003']
INTERVAL_SECONDS = 0.5

# Realistic value ranges for water quality monitoring
TEMP_RANGE = (20.0, 30.0)  # Celsius
CONDUCTIVITY_RANGE = (1000, 3000)  # microsiemens/cm


def generate_measurement(sensor_id):
    """Generate a realistic sensor measurement."""
    return {
        'sensor_id': sensor_id,
        'timestamp': datetime.utcnow().isoformat() + 'Z',
        'temperature': round(random.uniform(*TEMP_RANGE), 1),
        'conductivity': random.randint(*CONDUCTIVITY_RANGE)
    }


def send_measurement(measurement):
    """Send measurement to the API endpoint."""
    try:
        response = requests.post(API_URL, json=measurement, timeout=5)
        if response.status_code == 201:
            print(f"Sent: {measurement['sensor_id']} - "
                  f"Temp: {measurement['temperature']}°C, "
                  f"Cond: {measurement['conductivity']} µS/cm")
        else:
            print(f"Error {response.status_code}: {measurement['sensor_id']}")
    except requests.exceptions.ConnectionError:
        print(f"Connection failed for {measurement['sensor_id']} "
              f"(Is the server running at {API_URL}?)")
    except requests.exceptions.Timeout:
        print(f"Timeout for {measurement['sensor_id']}")
    except Exception as e:
        print(f"Unexpected error for {measurement['sensor_id']}: {e}")


def main():
    """Main loop - continuously send measurements from all sensors."""
    print(f"Starting sensor simulator...")
    print(f"Sending data to: {API_URL}")
```

```
    print(f"Simulating {len(SENSORS)} sensors")
    print(f"Interval: {INTERVAL_SECONDS} seconds")
    print("-" * 60)

    try:
        while True:
            for sensor_id in SENSORS:
                measurement = generate_measurement(sensor_id)
                send_measurement(measurement)

            time.sleep(INTERVAL_SECONDS)

    except KeyboardInterrupt:
        print("\n\nStopping simulator...")
        print("Goodbye!")


if __name__ == '__main__':
    main()
```