ASSIGNMENT 4

NAME: D. SAIMANIKANTA

REG NO: 192372348

COURSE:DATA BASE MANAGEMENT SYSTEM

CODE:CSA0593

SCENERIO:

Design a database for storing and processing environmental data generated by sensors in real-time. Requirements: Model tables to handle sensors, environmental readings, locations, and time-stamped data logs. Implement partitioning based on time intervals (e.g., daily) to efficiently manage large volumes of time-series data. Write queries to generate reports on environmental trends, detect anomalies in sensor readings, and visualize data by geographic area. Create views or materialized views for summarizing environmental data and improving performance of analytics queries

Environmental Data Storage and Processing System Design

To design a database for storing and processing environmental data generated by sensors in real-time, we will focus on managing sensor data, environmental readings, locations, and time-stamped logs. We will also implement partitioning to optimize performance when handling large volumes of time-series data. Additionally, we will create queries for generating environmental trend reports, detecting anomalies, and visualizing data by geographic areas.

1. Database Schema Design

The system will have the following main entities:

1. **Sensors**: Represents environmental sensors installed at various locations.
2. **Environmental Readings**: Stores individual sensor readings like temperature, humidity, air quality, etc.

3. **Locations**: Represents geographic areas or specific places where sensors are deployed.
4. **Data Logs**: Stores time-stamped logs for sensor readings and their status (for troubleshooting or analysis).
5. **Partitions**: Partitioning will be based on time intervals (e.g., daily) to manage time-series data efficiently.

Table 1: Sensors

This table stores metadata about each environmental sensor.

```
CREATE TABLE Sensors (
   sensor_id SERIAL PRIMARY KEY,
   sensor_name VARCHAR(255),
   sensor_type VARCHAR(255),    -- e.g., "temperature",
"humidity", "air_quality"
   unit VARCHAR(50),            -- e.g., "°C", "%", "ppm"
   location_id INT REFERENCES Locations(location_id),  --
Foreign key to Locations
   created_at TIMESTAMP DEFAULT
CURRENT_TIMESTAMP
);
```

Table 2: Locations

This table stores information about the geographic location of the sensors, such as city, region, or specific area.

sql

Copy code

```
REATE TABLE Locations (
    location_id SERIAL PRIMARY KEY,
    location_name VARCHAR(255),  -- e.g., "Downtown", "Forest Area"
    latitude DOUBLE PRECISION,
    longitude DOUBLE PRECISION,
    region VARCHAR(255),         -- e.g., "North Zone", "Southern Sector"
    description TEXT
```

Table 3: Environmental Readings

This table stores the actual environmental sensor readings, including data values such as temperature, humidity, and air qual

```
CREATE TABLE EnvironmentalReadings (
   reading_id SERIAL PRIMARY KEY,
   sensor_id INT REFERENCES Sensors(sensor_id),  -- Foreign key to Sensors
   timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   reading_value DOUBLE PRECISION,   -- The actual value recorded by the sensor
   status VARCHAR(50)
```

Table 4: Data Logs

This table stores logs of sensor readings, including time-stamped information on the status of the readings (used for troubleshooting or historical analysis).

```
CREATE TABLE DataLogs (
    log_id SERIAL PRIMARY KEY,
    reading_id INT REFERENCES EnvironmentalReadings(reading_id),  -- Foreign key to EnvironmentalReadings
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    log_message TEXT
);
```

2. Partitioning Strategy

The **EnvironmentalReadings** and **DataLogs** tables will be partitioned by time intervals (e.g., daily partitions). This will improve the performance of queries that filter by date ranges and will help in managing large volumes of time-series data.

Example of partitioning the **EnvironmentalReadings** table by day:

```
CREATE TABLE EnvironmentalReadings_2024_11_29 PARTITION OF
EnvironmentalReadings
FOR VALUES FROM ('2024-11-29') TO ('2024-11-30');
CREATE TABLE DataLogs_2024_11_29 PARTITION OF DataLogs
FOR VALUES FROM ('2024-11-29') TO ('2024-11-30');
```

This will allow queries to scan only the relevant partitions based on the date range.

Similarly, we can partition the **DataLogs** table by day:

3. SQL Queries for Reporting and Analytics

1. Environmental Trend Report

To track environmental trends (e.g., the number of readings for each sensor per day), we can aggregate data by sensor_id and timestamp.

```
ELECT sensor_id,

      DATE(timestamp) AS day,

      COUNT(*) AS readings_count

FROM EnvironmentalReadings

GROUP BY sensor_id, day

ORDER BY day DESC;
```

This query will give us the number of readings recorded for each sensor, grouped by day.

2. Anomaly Detection Query

To detect anomalies, we can define certain thresholds for sensor readings (e.g., temperature exceeding 40°C or air quality being too low). Here's an example query for detecting temperature anomalies:

```
SELECT s.sensor_id,

      r.timestamp,

      r.reading_value,

      s.sensor_type

FROM EnvironmentalReadings r

JOIN Sensors s ON r.sensor_id = s.sensor_id

WHERE s.sensor_type = 'temperature'

  AND (r.reading_value > 40 OR r.reading_value < -5);
```

wildetetemperature readings that are either too high (greater than 40°C) or too low (less than -5°C).

3. Visualize Data by Location

To visualize data by geographic area or location, we can calculate averages for each location, such as the average temperature:

```
SELECT l.location_name,
       AVG(r.reading_value) AS avg_temperature
FROM EnvironmentalReadings r
JOIN Sensors s ON r.sensor_id = s.sensor_id
JOIN Locations l ON s.location_id = l.location_id
WHERE s.sensor_type = 'temperature'
GROUP BY l.location_name
ORDER BY avg_temperature DESC;
```

This query provides the average temperature for each location (e.g., area, city) and orders them by temperature.

4. Environmental Performance Over Time

To monitor sensor performance over time (e.g., counting how many readings were recorded per sensor), we can use:

```
SELECT s.sensor_name,
    COUNT(r.reading_id) AS total_readings,
    DATE(r.timestamp) AS day
FROM EnvironmentalReadings r
JOIN Sensors s ON r.sensor_id = s.sensor_id
GROUP BY s.sensor_name, day
ORDER BY day DESC;
```

5. Conclusion

The system design includes the following key components:

- **Tables**: For sensors, locations, environmental readings, and data logs.
- **Partitioning**: The **EnvironmentalReadings** and **DataLogs** tables are partitioned by date to efficiently handle large volumes of time-series data.
- **Queries**: SQL queries for generating environmental trend reports, detecting anomalies, and visualizing data by location.
- **Materialized Views**: Precomputed views that summarize data, improving the performance of frequent analytics queries.

This design ensures that environmental data is efficiently stored, processed, and analyzed in real-time, while maintaining scalability and performance for large datasets. The partitioning and materialized views will significantly improve query performance when dealing with large volumes of time-series environmental data.