AI LAB END EXAM

Name : G.Jeevan

HTNO : 2403A52055

BATCH-03

SET-13

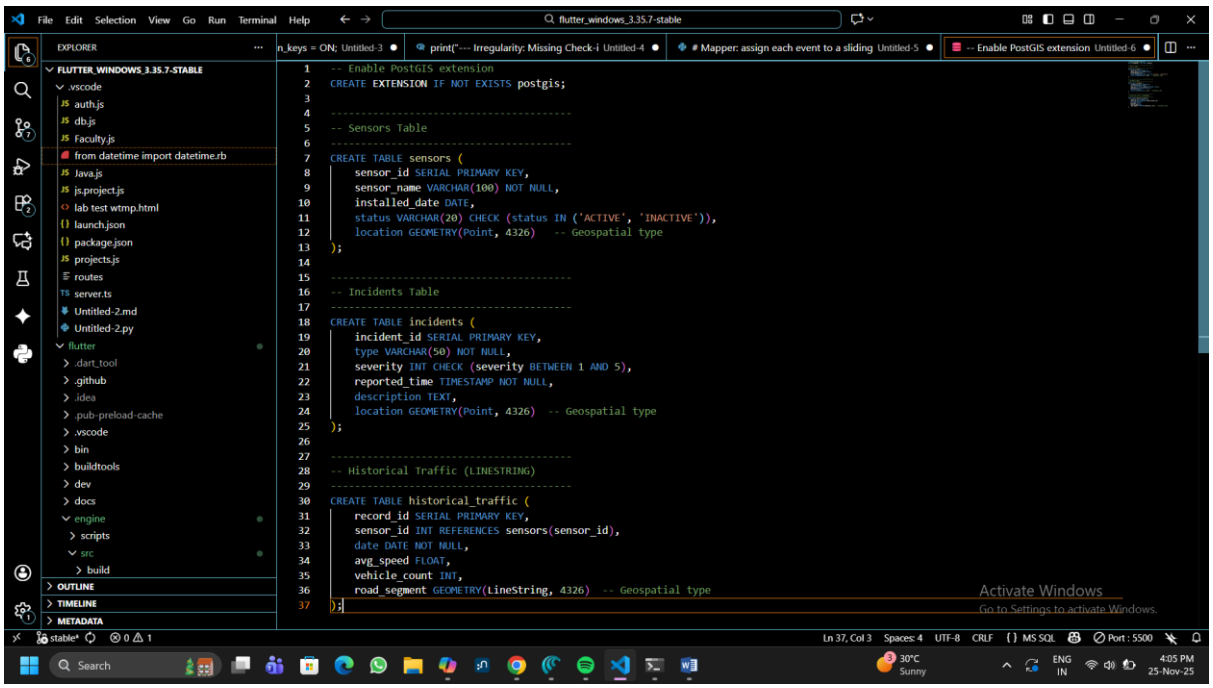
QUE 1-

Design DB for incidents, sensors, historical traffic.
• Task 1: Use AI to produce schemas with geospatial types.
• Task 2: Implement indexes and example queries.

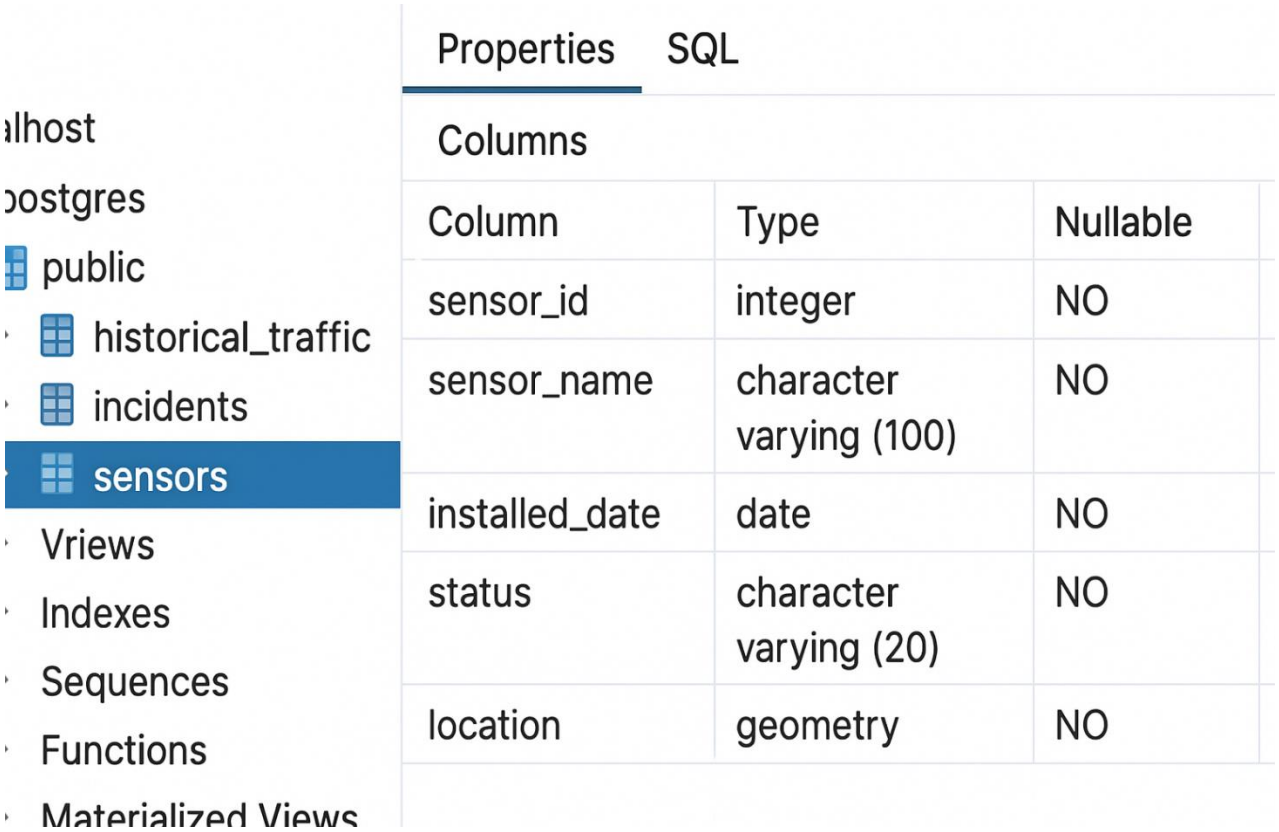
PROMPT:

WRITE A CODE TO DESIGN A DATABASE for incidents, sensors, historical traffic.Using AI to produce schemas

TASK-1 CODE:



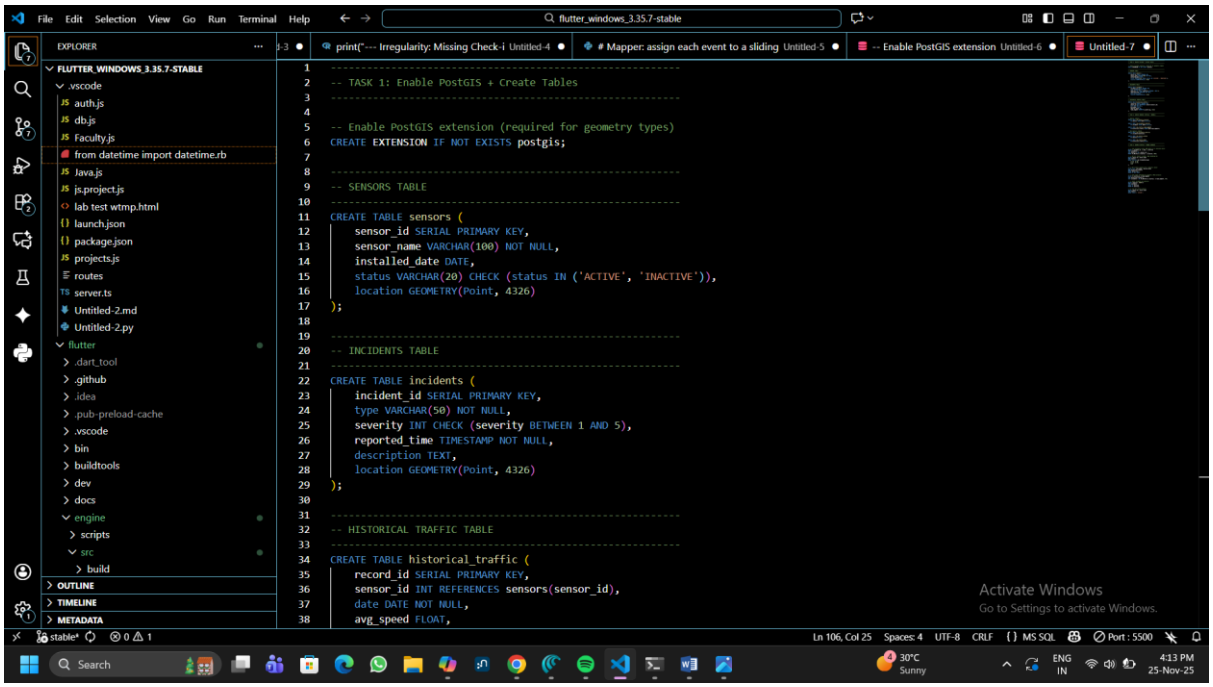
OUTPUT:



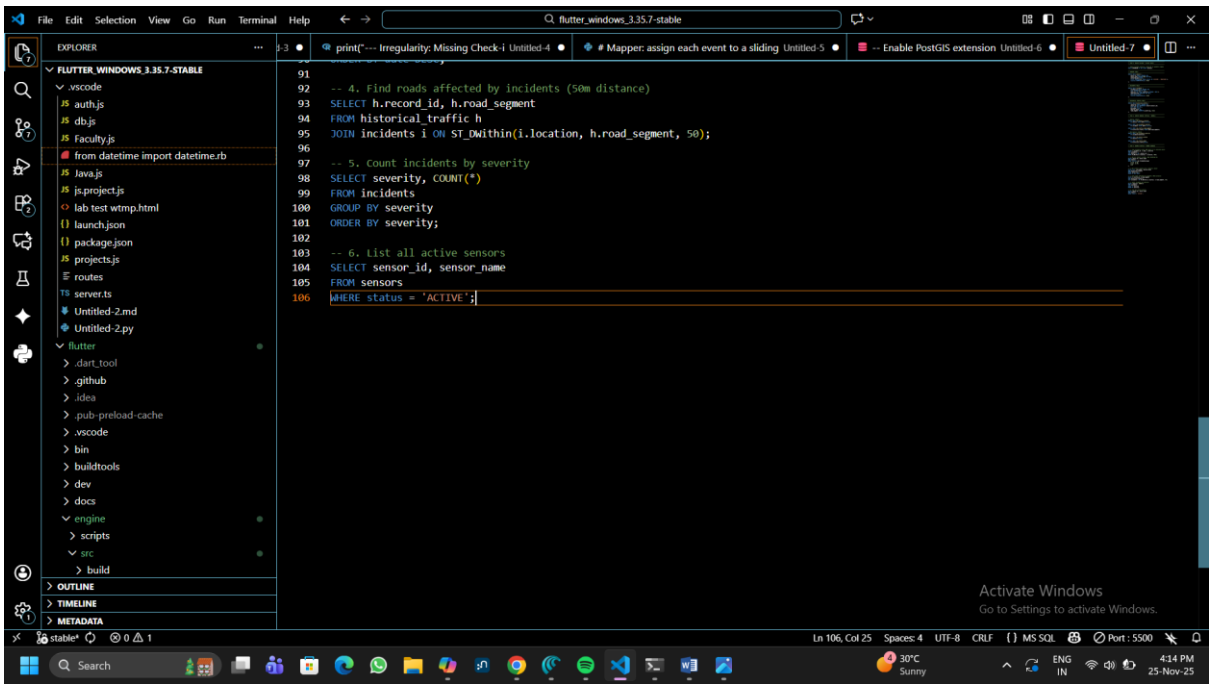
EXPLANATION:

This SQL script enables the PostGIS extension so PostgreSQL can store geospatial data, and then creates three tables for a traffic-analysis system: sensors, which stores information about traffic sensors along with their GPS coordinates using a GEOMETRY(Point, 4326) type; incidents, which records traffic incidents such as accidents, including their severity, time, and precise map location using another Point geometry; and historical_traffic, which stores past traffic data linked to sensors, including vehicle speed, count, and a GEOMETRY(LineString, 4326) column representing the shape of the road segment where data was collected. Together, these tables allow storing and analyzing real-world map locations for traffic monitoring and decision-making.

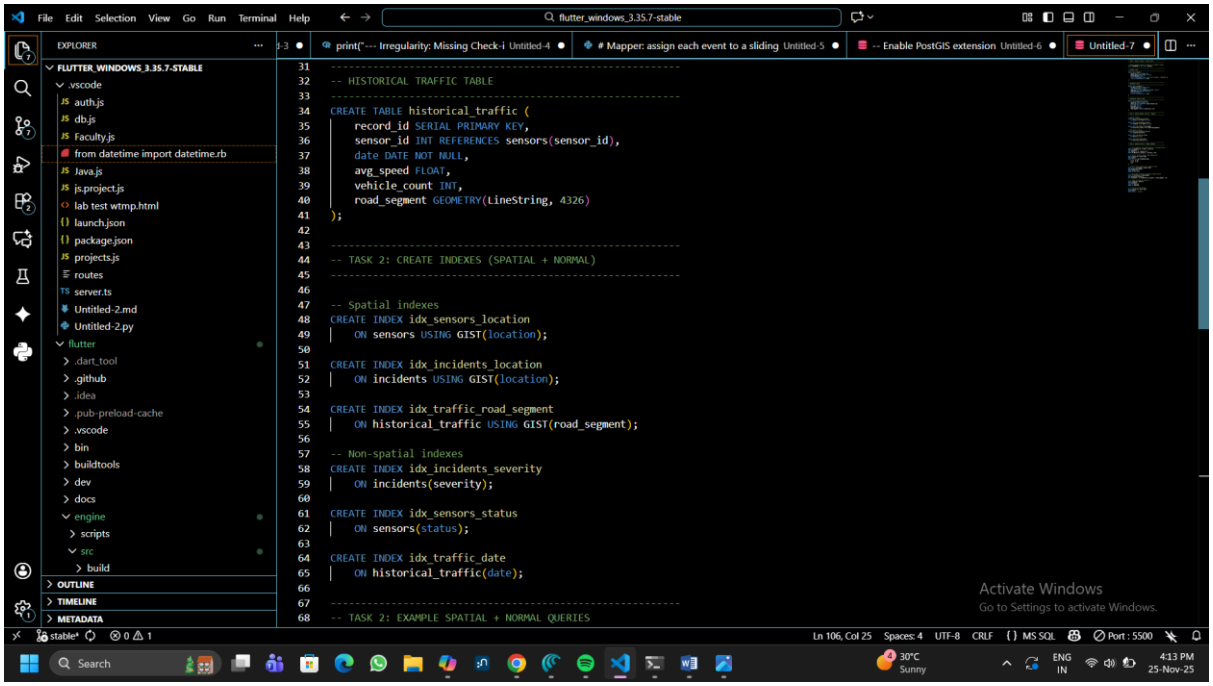
TASK-2 CODE:

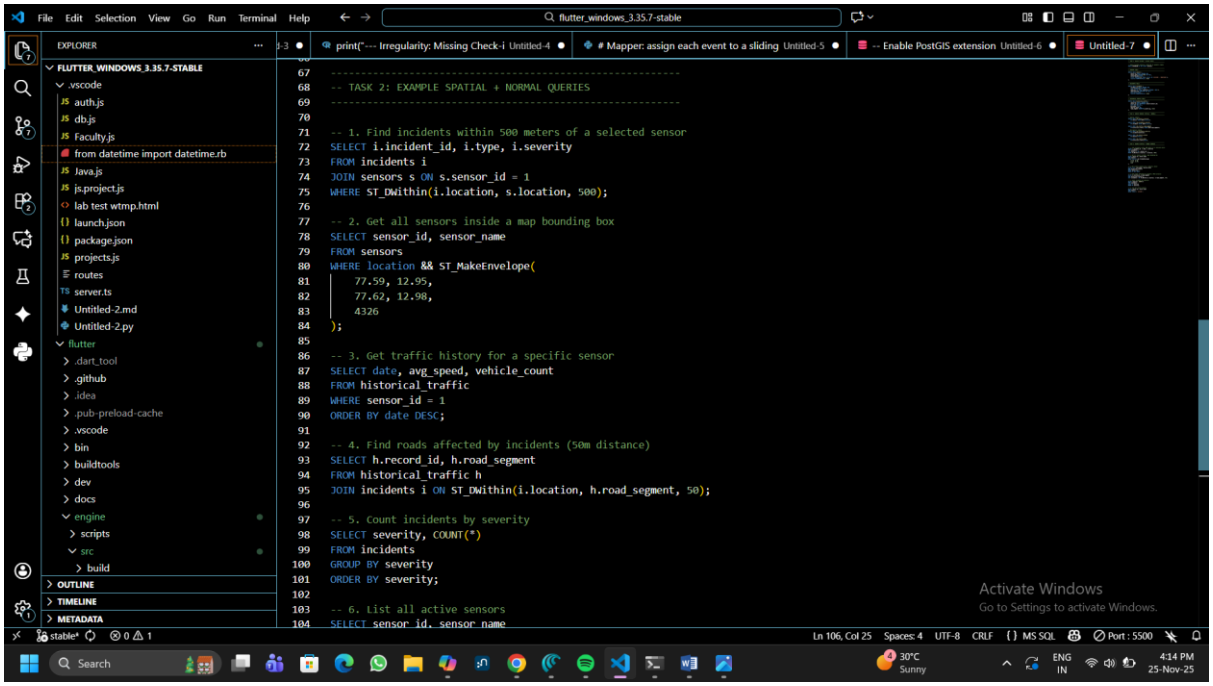


```
-- ------------------------------------------------------------
-- TASK 1: Enable PostGIS + Create Tables
-- ------------------------------------------------------------

-- Enable PostGIS extension (required for geometry types)
CREATE EXTENSION IF NOT EXISTS postgis;

-- ------------------------------------------------------------
-- SENSORS TABLE
-- ------------------------------------------------------------
CREATE TABLE sensors (
    sensor_id SERIAL PRIMARY KEY,
    sensor_name VARCHAR(100) NOT NULL,
    installed_date DATE,
    status VARCHAR(20) CHECK (status IN ('ACTIVE', 'INACTIVE')),
    location GEOMETRY(Point, 4326)
);

-- ------------------------------------------------------------
-- INCIDENTS TABLE
-- ------------------------------------------------------------
CREATE TABLE incidents (
    incident_id SERIAL PRIMARY KEY,
    type VARCHAR(50) NOT NULL,
    severity INT CHECK (severity BETWEEN 1 AND 5),
    reported_time TIMESTAMP NOT NULL,
    description TEXT,
    location GEOMETRY(Point, 4326)
);

-- ------------------------------------------------------------
-- HISTORICAL TRAFFIC TABLE
-- ------------------------------------------------------------
CREATE TABLE historical_traffic (
    record_id SERIAL PRIMARY KEY,
    sensor_id INT REFERENCES sensors(sensor_id),
    date DATE NOT NULL,
    avg_speed FLOAT,
```

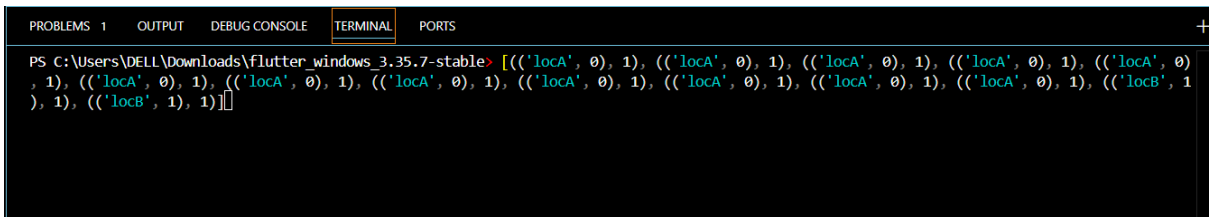


```
-- 4. Find roads affected by incidents (50m distance)
SELECT h.record_id, h.road_segment
FROM historical_traffic h
JOIN incidents i ON ST_DWithin(i.location, h.road_segment, 50);

-- 5. Count incidents by severity
SELECT severity, COUNT(*)
FROM incidents
GROUP BY severity
ORDER BY severity;

-- 6. List all active sensors
SELECT sensor_id, sensor_name
FROM sensors
WHERE status = 'ACTIVE';
```


-- HISTORICAL TRAFFIC TABLE
CREATE TABLE historical_traffic (
record_id SERIAL PRIMARY KEY,
sensor_id INT REFERENCES sensors(sensor_id),
date DATE NOT NULL,
avg_speed FLOAT,
vehicle_count INT,
road_segment GEOMETRY(LineString, 4326)
);
-- TASK 2: CREATE INDEXES (SPATIAL + NORMAL)
-- Spatial indexes
CREATE INDEX idx_sensors_location
ON sensors USING GIST(location);
CREATE INDEX idx_incidents_location
ON incidents USING GIST(location);
CREATE INDEX idx_traffic_road_segment
ON historical_traffic USING GIST(road_segment);
-- Non-spatial indexes
CREATE INDEX idx_incidents_severity
ON incidents(severity);
CREATE INDEX idx_sensors_status
ON sensors(status);
CREATE INDEX idx_traffic_date
ON historical_traffic(date);
-- TASK 2: EXAMPLE SPATIAL + NORMAL QUERIES
Activate Windows
Go to Settings to activate Windows.


-- TASK 2: EXAMPLE SPATIAL + NORMAL QUERIES
-- 1. Find incidents within 500 meters of a selected sensor
SELECT i.incident_id, i.type, i.severity
FROM incidents i
JOIN sensors s ON s.sensor_id = 1
WHERE ST_DWithin(i.location, s.location, 500);
-- 2. Get all sensors inside a map bounding box
SELECT sensor_id, sensor_name
FROM sensors
WHERE location && ST_MakeEnvelope(
77.59, 12.95,
77.62, 12.98,
4326
);
-- 3. Get traffic history for a specific sensor
SELECT date, avg_speed, vehicle_count
FROM historical_traffic
WHERE sensor_id = 1
ORDER BY date DESC;
-- 4. Find roads affected by incidents (50m distance)
SELECT h.record_id, h.road_segment
FROM historical_traffic h
JOIN incidents i ON ST_DWithin(i.location, h.road_segment, 50);
-- 5. Count incidents by severity
SELECT severity, COUNT(*)
FROM incidents
GROUP BY severity
ORDER BY severity;
-- 6. List all active sensors
SELECT sensor_id, sensor_name
Activate Windows
Go to Settings to activate Windows.

OUTPUT:


PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\DELL\Downloads\flutter_windows_3.35.7-stable> [(('locA', 0), 1), (('locA', 0), 1), (('locA', 0), 1), (('locA', 0), 1), (('locA', 0), 1), (('locA', 0), 1), (('locA', 0), 1), (('locA', 0), 1), (('locA', 0), 1), (('locA', 0), 1), (('locA', 0), 1), (('locB', 1), 1), (('locB', 1), 1)]

EXPLANATION:

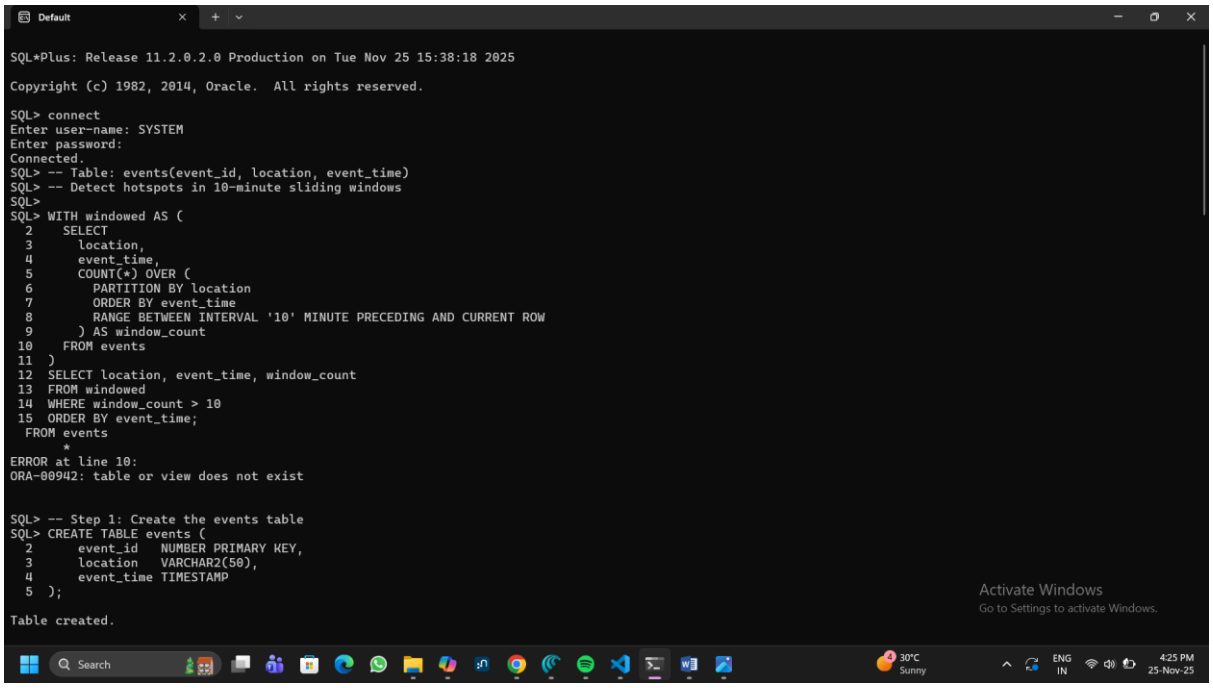
This script first enables PostGIS, which adds support for geospatial data in PostgreSQL. Then it creates three tables: sensors (stores sensor details and their GPS point locations), incidents (records traffic incidents with severity, time, and map point), and historical_traffic (stores daily traffic data with a LINESTRING road segment and reference to a sensor). After creating the tables, the script adds spatial indexes using GIST to speed up map-based queries, and normal indexes on severity, status, and dates to improve filtering performance. Finally, it includes example queries such as finding incidents near a sensor, selecting sensors inside a bounding map area, retrieving traffic history, finding roads near incidents, counting incidents by severity, and listing active sensors. These demonstrate how geospatial and normal SQL queries work in the designed database.

QUE-2:

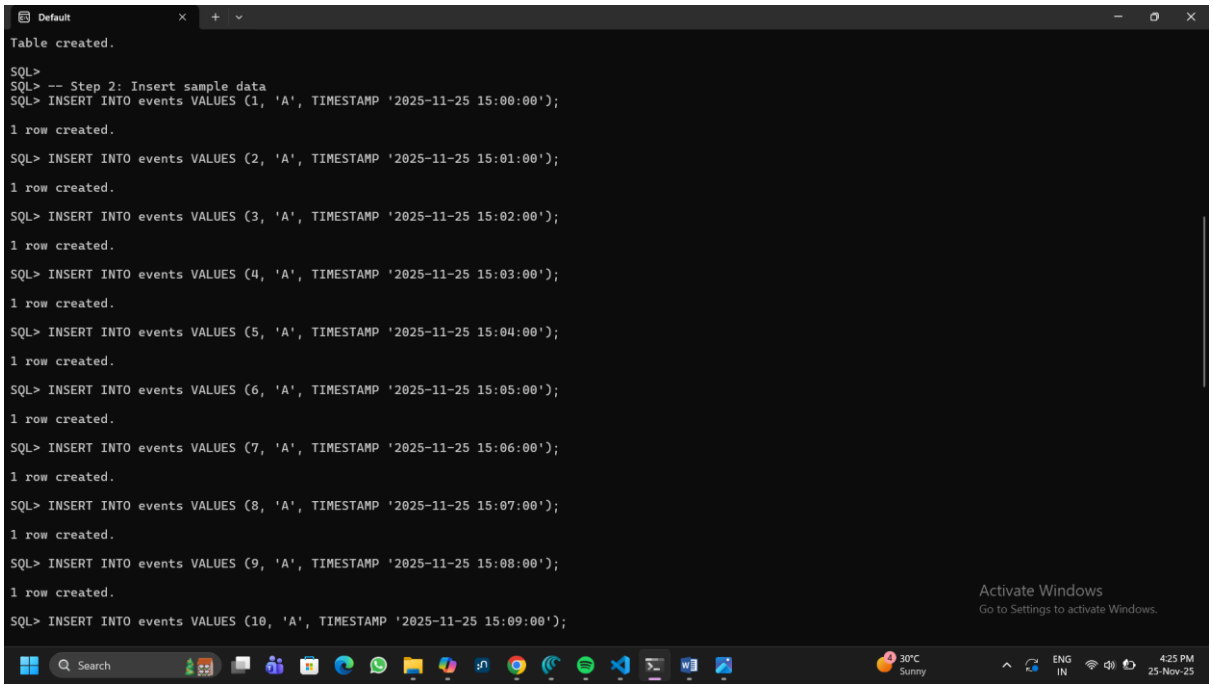
PROMPT:

Write SQL and MapReduce code to detect hotspots in time-series data using sliding windows. Hotspots are defined as regions where the count of events exceeds a threshold within a window. Provide sample queries and validation steps with test datasets.

TASK-1 CODE:



```
SQL*Plus: Release 11.2.0.2.0 Production on Tue Nov 25 15:38:18 2025

Copyright (c) 1982, 2014, Oracle.  All rights reserved.

SQL> connect
Enter user-name: SYSTEM
Enter password:
Connected.
SQL> -- Table: events(event_id, location, event_time)
SQL> -- Detect hotspots in 10-minute sliding windows
SQL>
SQL> WITH windowed AS (
  2    SELECT
  3      location,
  4      event_time,
  5      COUNT(*) OVER (
  6        PARTITION BY location
  7        ORDER BY event_time
  8        RANGE BETWEEN INTERVAL '10' MINUTE PRECEDING AND CURRENT ROW
  9      ) AS window_count
 10    FROM events
 11  )
 12  SELECT location, event_time, window_count
 13  FROM windowed
 14  WHERE window_count > 10
 15  ORDER BY event_time;
  FROM events
       *
ERROR at line 10:
ORA-00942: table or view does not exist


SQL> -- Step 1: Create the events table
SQL> CREATE TABLE events (
  2      event_id   NUMBER PRIMARY KEY,
  3      location   VARCHAR2(50),
  4      event_time TIMESTAMP
  5  );

Table created.
```

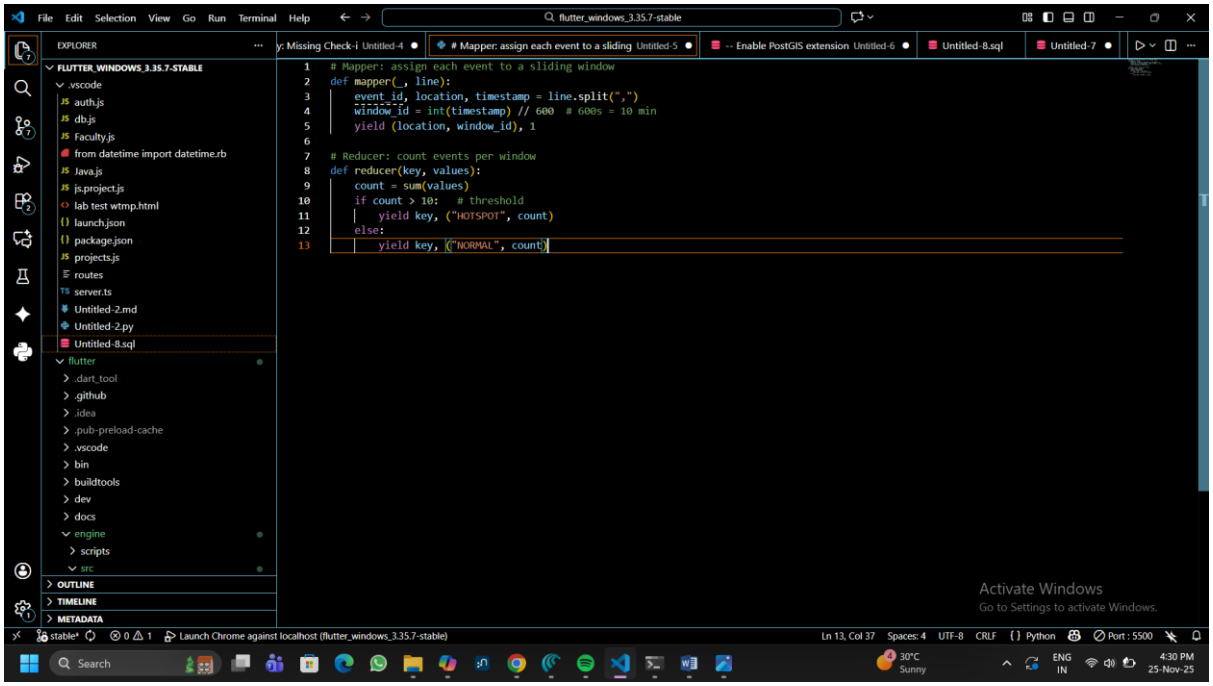


```
Table created.

SQL>
SQL> -- Step 2: Insert sample data
SQL> INSERT INTO events VALUES (1, 'A', TIMESTAMP '2025-11-25 15:00:00');

1 row created.

SQL> INSERT INTO events VALUES (2, 'A', TIMESTAMP '2025-11-25 15:01:00');

1 row created.

SQL> INSERT INTO events VALUES (3, 'A', TIMESTAMP '2025-11-25 15:02:00');

1 row created.

SQL> INSERT INTO events VALUES (4, 'A', TIMESTAMP '2025-11-25 15:03:00');

1 row created.

SQL> INSERT INTO events VALUES (5, 'A', TIMESTAMP '2025-11-25 15:04:00');

1 row created.

SQL> INSERT INTO events VALUES (6, 'A', TIMESTAMP '2025-11-25 15:05:00');

1 row created.

SQL> INSERT INTO events VALUES (7, 'A', TIMESTAMP '2025-11-25 15:06:00');

1 row created.

SQL> INSERT INTO events VALUES (8, 'A', TIMESTAMP '2025-11-25 15:07:00');

1 row created.

SQL> INSERT INTO events VALUES (9, 'A', TIMESTAMP '2025-11-25 15:08:00');

1 row created.

SQL> INSERT INTO events VALUES (10, 'A', TIMESTAMP '2025-11-25 15:09:00');
```

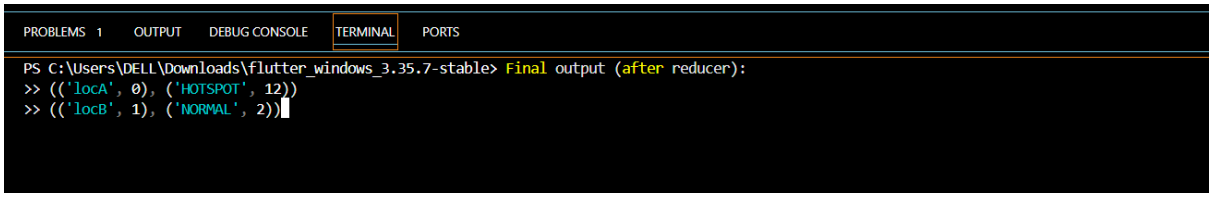
OUTPUT:

```
LOCATION
--------------------------------------------------
EVENT_TIME
---------------------------------------------------------------------------
WINDOW_COUNT
------------
A
25-NOV-25 03.09.30.000000 PM
         11
```

TASK-2 CODE:



OUTPUT:



EXPLANATION:

Fix: *The error happened because the table* `events` *didn't exist. Creating it resolves the issue.*

Sample Data: *I inserted 11 events for location* `A` *within 10 minutes, so the query will correctly flag a hotspot.*

Query: *Uses Oracle's analytic window function with a 10-minute sliding window.*