Project 567
Sophia Efthymia Sideri
Panagiotis Karouzakis

Problem:

We want to find the best route between two or more points on the map to lower air pollution and achieve optimal pricing. We will also include user's preferences on the UI for this decision.

Datasets:

https://www.fueleconomy.gov/feg/EPAGreenGuide/Smartway/pdf/SmartWay%20Vehicle%20List%20MY%202010.pdf
https://www.fueleconomy.gov/feg/EPAGreenGuide/Smartway/pdf/SmartWay%20Vehicle%20List%20for%20MY%202025.pdf
We will use 50 instances from each dataset (50 from the dataset of 2025, and 50 from 2010) for the first demonstration, and we will not use all the columns. We will provide only the information needed to calculate the **optimal routing** according to the 1) ***user's preferences*** (they would be able to choose trans, vehicle class and drive), 2) ***air pollution score***, 3) ***approximate fuel calculation***, 4) ***price of the vehicle***.

Schema:

1. **CAR**
    - id: Unique identifier for the car.
    - model: Car model name.
    - displ: Engine volume in liters.
    - cylinders: Engine combustion chambers count.
    - trans: Transmission type (e.g., Automatic, Manual).
    - drive: Drive type (e.g., AWD, FWD, RWD).
    - fuel: Fuel type (e.g., Gasoline, Electric).
    - Cert_region: Region for emissions compliance.
    - stnd: Emissions standard identifier.
    - stnd_description: Human-readable emissions standard identifier.
    - Underhood_id: Engine's unique identifier.
    - veh_class: Vehicle class (e.g., SUV, Sedan).
    - air_pollution_score: Air pollution score (numeric).
    - city_mpg: City miles per gallon.
    - hwy_mpg: Highway miles per gallon.
    - cmb_mpg: Combined miles per gallon.
    - greenhouse_gas_score: Greenhouse gas score (numeric).
    - smartway: Boolean indicating if the car is SmartWay certified.
    - price: Price of the vehicle (numeric - we will put it randomly).
2. **ADDRESS**

- `id`: Unique identifier for the address.
- `name`: Name or description of the location.
- `point`: Geospatial point (e.g., latitude, longitude).
- `color`: Color for pinpoint.

3. **`ROUTE`**
   - `id`: Unique identifier for the route.
   - `start_point`: Reference to the starting `ADDRESS` node.
   - `end_point`: Reference to the ending `ADDRESS` node.
   - `geometry`: Polyline geometry for the route (from the API).
   - `distance`: Total distance of the route (in meters).
   - `duration`: Total duration of the route (in seconds).
   - `steps`: Optional: Array of step-by-step instructions (if required).

Relationships

1. **`TRAVELS_TO`**
   - **From:** `ADDRESS`
   - **To:** `ADDRESS`
   - **Properties:**
     i. `route_id`: Reference to the associated `ROUTE`.
     ii. `distance`: Distance of the route (from OpenRouteService).
     iii. `duration`: Time taken for the route (from OpenRouteService).
     iv. `geometry`: Polyline geometry (optional, stored here if not in `ROUTE`).
     v. `instructions`: Optional: Step-by-step instructions (only if necessary).
2. **`OPTIMAL_ROUTE`**
   - **From:** `CAR`
   - **To:** `ROUTE`
   - **Properties:**
     i. `priority`: Optimization priority (e.g., `low_emission`, `shortest_time`, etc.).
     ii. `total_distance`: Total distance of the assigned route.
     iii. `total_duration`: Total time for the assigned route.
     iv. `fuel_estimate`: Estimated fuel consumption for the route (calculated based on car attributes, the less the better).
     v. `polution_score_estimate`: Estimated pollution score for the route (calculated based on car attributes, the bigger the score the better).
3. **`HAS_ROUTE`**
   - **From:** `CAR`
   - **To:** `ADDRESS` (via `ROUTE` intermediary).
   - **Description:** Connect cars to their respective routes and waypoints.

- **Properties:**
    1. sequence: Order of visits for the car (e.g., waypoint 1, waypoint 2).

CYPHER QUERIES

1. Create CAR nodes

```
CREATE (c:CAR {
    id: $id,
    model: $model,
    displ: $displ
    cylinders: $cylinders
    trans: $trans,
    drive: $drive,
    fuel: $fuel,
    cert_region: $cert_region
    stnd: $stnd,
    stnd_description: $stnd_description,
    underhood_id: $underhood_id,
    veh_class: $veh_class,
    air_pollution_score: $air_pollution_score,
    city_mpg: $city_mpg,
    hwy_mpg: $hwy_mpg,
    cmb_mpg: $cmb_mpg,
    greenhouse_gas_score: $greenhouse_gas_score,
    smartway: $smartway,
    price: $price
})
```

2. Create ADDRESS nodes

```
CREATE (a:ADDRESS {
    id: $id,
    name: $name,
    point: point({latitude: $latitude, longitude: $longitude}),
    color: $color
})
```

3. Create `ROUTE` nodes

```
CREATE (r:ROUTE {
    id: $id,
    start_point: $start_point_id,
    end_point: $end_point_id,
    geometry: $geometry,
    distance: $distance,
    duration: $duration,
    steps: $steps
})
```

4. Create `TRAVELS_TO` relationships

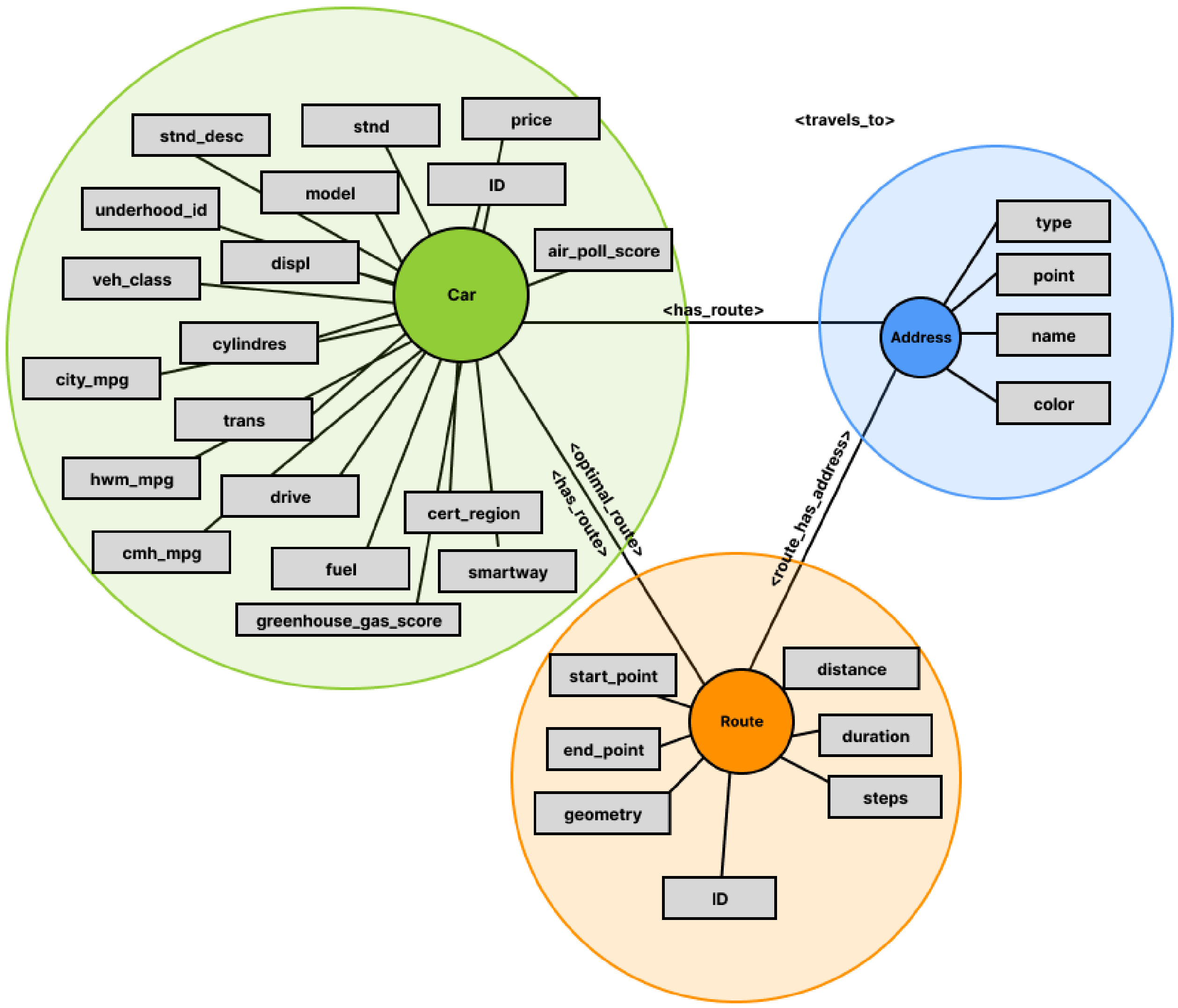
```
MATCH (a1:ADDRESS {id: $start_id}), (a2:ADDRESS {id: $end_id})
CREATE (a1)-[:TRAVELS_TO {
    route_id: $route_id,
    distance: $distance,
    duration: $duration,
    geometry: $geometry,
    instructions: $instructions
}]->(a2)
```

5. Create `OPTIMAL_ROUTE` relationships

```
MATCH (c:CAR {id: $car_id}), (r:ROUTE {id: $route_id})
CREATE (c)-[:OPTIMAL_ROUTE {
    priority: $priority,
    total_distance: $total_distance,
    total_duration: $total_duration,
    fuel_estimate: $fuel_estimate,
    pollution_score_estimate: $pollution_score_estimate
}]->(r)
```

6. Create HAS_ROUTE relationships

```
MATCH (c:CAR {id: $car_id}), (a:ADDRESS {id: $address_id}), (r:ROUTE
{id: $route_id})
CREATE (c)-[:HAS_ROUTE {
    sequence: $sequence
}]->(a)
```



Example Queries

1.Get all cars with automatic or semi-automatic transmission

```
MATCH (car)
WHERE car.trans CONTAINS "Auto"
RETURN car.model AS Model, car.trans AS Transmission
```

2. Get the 5 cars with the lowest air pollution score and price

```
MATCH (car)
WHERE car.price IS NOT NULL AND car.air_pollution_score IS NOT NULL
RETURN car.model AS Model, car.price AS Price, car.air_pollution_score
AS AirPollutionScore
ORDER BY car.air_pollution_score ASC, car.price ASC
LIMIT 5
```

3. Get the 5 cars with the best greenhouse gas score and highest price

```
MATCH (car)
WHERE car.price IS NOT NULL AND car.greenhouse_gas_score IS NOT NULL
RETURN car.model AS Model, car.price AS Price,
car.greenhouse_gas_score AS GreenhouseGasScore
ORDER BY car.greenhouse_gas_score DESC, car.price ASC
LIMIT 5
```

4. Get the 10 cars with lower air pollution scores, higher City MPG, and lower prices

```
MATCH (car)
WHERE car.air_pollution_score IS NOT NULL AND car.city_mpg IS NOT NULL
AND car.price IS NOT NULL
RETURN car.model AS Model, car.air_pollution_score AS
AirPollutionScore, car.city_mpg AS CityMPG, car.price AS Price
ORDER BY car.air_pollution_score ASC, car.city_mpg DESC, car.price ASC
LIMIT 10
```

5.Get the best Audi with the lowest air pollution score and highest Combined MPG

```
MATCH (car)
```

```
WHERE car.model CONTAINS "AUDI" AND car.air_pollution_score IS NOT
NULL AND car.cmb_mpg IS NOT NULL
RETURN car.model AS Model, car.air_pollution_score AS
AirPollutionScore, car.cmb_mpg AS CombinedMPG
ORDER BY car.air_pollution_score ASC, car.cmb_mpg DESC
LIMIT 1
```

6. Optimal Car for a Route (the best car for a specific route based on air pollution score, fuel estimate, and user preferences)

```
MATCH (car:CAR), (route:ROUTE {id: $route_id})
WHERE car.trans CONTAINS $trans
  AND car.drive = $drive
  AND car.fuel = $fuel
  AND car.veh_class = $veh_class
WITH car, route,
     car.air_pollution_score AS pollutionScore,
     (route.distance / car.cmb_mpg) AS fuelEstimate
ORDER BY pollutionScore ASC, fuelEstimate ASC
LIMIT 1
RETURN car.model AS OptimalCar, pollutionScore, fuelEstimate,
route.distance AS Distance
```

7. Optimal Car and Route for Given Addresses (Find the best car and route for traveling between two points (start and end addresses)).

```
MATCH (car:CAR), (start:ADDRESS {id: $start_id}), (end:ADDRESS {id:
$end_id})
MATCH (start)-[:TRAVELS_TO {route_id: route.id}]->(end)
WITH car, route,
     car.air_pollution_score AS pollutionScore,
     (route.distance / car.cmb_mpg) AS fuelEstimate,
     route.distance AS routeDistance,
     route.duration AS routeDuration
ORDER BY pollutionScore ASC, fuelEstimate ASC, routeDistance ASC
LIMIT 1
RETURN car.model AS OptimalCar,
       route.id AS OptimalRoute,
       pollutionScore,
       fuelEstimate,
       routeDistance,
       routeDuration
```

8. Optimal Car and Route Based on User Preferences (Find an optimal car and route considering user-defined preferences for route priority (low_emission, shortest_time, etc.).))

```
MATCH (car:CAR), (route:ROUTE)
WHERE car.trans = $trans
  AND car.drive = $drive
  AND car.fuel = $fuel
  AND car.veh_class = $veh_class
WITH car, route,
     CASE $priority
         WHEN 'low_emission' THEN car.air_pollution_score
         WHEN 'shortest_time' THEN route.duration
         ELSE (route.distance / car.cmb_mpg)
     END AS optimizationScore
ORDER BY optimizationScore ASC
LIMIT 1
RETURN car.model AS OptimalCar,
       route.id AS OptimalRoute,
       car.air_pollution_score AS PollutionScore,
       route.duration AS Duration,
       (route.distance / car.cmb_mpg) AS FuelEstimate
```

Things to do:

- ☐ **UI:**
  - ☐ When creating a Vehicle, the user would be able to select the vehicle he wants through search & filters for transmission (dropdown selection), drive type(dropdown selection), fuel type (dropdown selection), vehicle class(dropdown selection), air pollution score threshold (range), smart way (a check), and the price of the car (range). So the user either will search for the model of the car immediately or use filters to do a search an then select a vehicle.
  - ☐ When receiving a result, show the optimal route and how much pollution score we have? maybe also the gas cost ? // we will see

    - [ ] Have a save button on the results in order to save the routes
    - [ ] If we have time, we can make a modal to create a new car instance
- [ ] **ASP:**
    - [ ] Need to create the priority rules
- [ ] **DATA**:
    - [x] ~~Make a small python code that will read 50 random rows of the dataset and will insert in our db (with a cypher query) the columns we want.~~
    - [x] ~~Do a running script to load the data initially~~
    - [x] ~~Load data in Neo4j.~~
- [ ] **Python:**
    - [ ] handle cypher queries
        - [ ] Normalization of the characters (α, β, γ... ä, ö, ü, ß, etc) - but I think we can pass the id’s rather than the name of the address
    - [ ] connect UI with the ASP logic
    - [x] ~~Load data on init~~