**NE04j**

**Online Neo4j sandbox temporary ex.**

**1. Query to create database with player, coach, team details along with their relationships:**

cypher

Copy code

CREATE

(russell:PLAYER {name: "Russell Westbrook", age: 33, number: 0, height: 1.91, weight: 91}),

(lebron:PLAYER {name: "LeBron James", age: 36, number: 6, height: 2.06, weight: 113}),

(anthony:PLAYER {name: "Anthony Davis", age: 28, number: 23, height: 2.08, weight: 115}),

(ja:PLAYER {name: "Ja Morant", age: 22, number: 12, height: 1.91, weight: 79}),

(frank:COACH {name: "Frank Vogel"}),

(taylor:COACH {name: "Taylor Jenkins"}),

(lakers:TEAM {name: "LA Lakers"}),

(mavericks:TEAM {name: "Dallas Mavericks"}),

(lebron)-[:TEAMMATES]->(russell),

(lebron)<-[:TEAMMATES]-(russell),

(ja)-[:TEAMMATES]->(anthony),

(ja)<-[:TEAMMATES]-(anthony),

(frank)-[:COACHES]->(lebron),

(frank)-[:COACHES]->(russell),

(taylor)-[:COACHES]->(ja),

(taylor)-[:COACHES]->(anthony),

(lebron)-[:PLAYS\_FOR {salary: 40000000}]->(lakers),

(russell)-[:PLAYS\_FOR {salary: 33000000}]->(lakers),

(anthony)-[:PLAYS\_FOR {salary: 38000000}]->(mavericks),

(ja)-[:PLAYS\_FOR {salary: 8000000}]->(mavericks),

(frank)-[:COACHES\_FOR]->(lakers),

(taylor)-[:COACHES\_FOR]->(mavericks),

(lebron)-[:PLAYED\_AGAINST {minutes: 38, points: 32, assists: 6, rebounds: 6, turnovers: 2}]->(mavericks),

(russell)-[:PLAYED\_AGAINST {minutes: 29, points: 16, assists: 12, rebounds: 11, turnovers: 16}]->(mavericks),

(anthony)-[:PLAYED\_AGAINST {minutes: 36, points: 27, assists: 2, rebounds: 8, turnovers: 1}]->(lakers),

(ja)-[:PLAYED\_AGAINST {minutes: 43, points: 42, assists: 7, rebounds: 8, turnovers: 4}]->(lakers);

**2. Query to fetch the whole graph model with all the nodes and relationships:**

cypher

Copy code

MATCH (n) RETURN n;

**3. Query to fetch only the player nodes:**

cypher

Copy code

MATCH (player:PLAYER) RETURN player;

**4. Query to fetch only the player names:**

cypher

Copy code

MATCH (player:PLAYER) RETURN player.name;

**5. Query to fetch only the player names and their height values:**

cypher

Copy code

MATCH (player:PLAYER) RETURN player.name, player.height;

**6. Query to fetch player names with their height values and rename the column headers:**

cypher

Copy code

MATCH (player:PLAYER) RETURN player.name AS NAME, player.height AS HEIGHT;

**7. Query to fetch a specific player node using the name attribute:**

cypher

Copy code

MATCH (player:PLAYER) WHERE player.name = 'Anthony Davis' RETURN player;

**8. Alternate method to fetch a specific player node using the name attribute:**

cypher

Copy code

MATCH (player:PLAYER {name: "Anthony Davis"}) RETURN player;

**9. Query to fetch all player nodes except a specific player:**

cypher

Copy code

MATCH (player:PLAYER) WHERE player.name <> 'Anthony Davis' RETURN player;

**10. Query to fetch all player nodes whose height is greater than the specified value:**

cypher

Copy code

MATCH (player:PLAYER) WHERE player.height >= 2 RETURN player;

**11. Query to fetch all player nodes whose BMI is greater than the specified value:**

cypher

Copy code

MATCH (player:PLAYER) WHERE (player.weight / (player.height \* player.height)) > 25 RETURN player;

**12. Query to fetch players whose weight is greater than 100 and height is greater than 2:**

cypher

Copy code

MATCH (player:PLAYER) WHERE player.weight >= 100 AND player.height > 2 RETURN player;

**13. Query to fetch players whose weight is greater than 100 or height is greater than 2:**

cypher

Copy code

MATCH (player:PLAYER) WHERE player.weight >= 100 OR player.height > 2 RETURN player;

**14. Query to fetch players who do not satisfy the weight and height conditions using NOT:**

cypher

Copy code

MATCH (player:PLAYER) WHERE NOT (player.weight >= 100 AND player.height > 2) RETURN player;

**15. Query to fetch players not satisfying conditions, limit results:**

cypher

Copy code

MATCH (player:PLAYER) WHERE NOT (player.weight >= 100 AND player.height > 2) RETURN player LIMIT 1;

**16. Query to skip the first two players and limit results:**

cypher

Copy code

MATCH (player:PLAYER) RETURN player SKIP 2 LIMIT 2;

**17. Query to display players based on height in descending order:**

cypher

Copy code

MATCH (player:PLAYER) RETURN player ORDER BY player.height DESC;

**18. Query to display players based on height in ascending order:**

cypher

Copy code

MATCH (player:PLAYER) RETURN player ORDER BY player.height ASC;

**19. Query to display all PLAYER and COACH nodes:**

cypher

Copy code

MATCH (player:PLAYER), (coach:COACH) RETURN player, coach;

**20. Query to display players who play for the team "LA Lakers":**

cypher

Copy code

MATCH (player:PLAYER)-[:PLAYS\_FOR]->(team:TEAM) WHERE team.name = "LA Lakers" RETURN player, team;

**21. Query to display players who play for "LA Lakers" or "Dallas Mavericks":**

cypher

Copy code

MATCH (player:PLAYER)-[:PLAYS\_FOR]->(team:TEAM) WHERE team.name = "LA Lakers" OR team.name = "Dallas Mavericks" RETURN player, team;

**22. Query to display players whose salary is greater than 35000000:**

cypher

Copy code

MATCH (player:PLAYER)-[contract:PLAYS\_FOR]->(team:TEAM) WHERE contract.salary > 35000000 RETURN player;

**23. Query to display teammates of "LeBron James":**

cypher

Copy code

MATCH (player:PLAYER {name: "LeBron James"})-[:TEAMMATES]->(teammate:PLAYER) RETURN teammate;

**24. Query to display teammates of "LeBron James" whose salary is greater than 20000000:**

cypher

Copy code

MATCH (player:PLAYER {name: "LeBron James"})-[:TEAMMATES]->(teammate:PLAYER)

MATCH (teammate)-[contract:PLAYS\_FOR]->(team:TEAM)

WHERE contract.salary >= 20000000

RETURN teammate;

**25. Query to display the number of games played by each player:**

cypher

Copy code

MATCH (player:PLAYER)-[gamePlayed:PLAYED\_AGAINST]->(team:TEAM) RETURN player.name, COUNT(gamePlayed);

**26. Query to display players by average points scored in descending order:**

cypher

Copy code

MATCH (player:PLAYER)-[gamePlayed:PLAYED\_AGAINST]->(team:TEAM)

RETURN player.name, AVG(gamePlayed.points) AS TOP

ORDER BY TOP DESC;

**27. Query to delete a specific player node:**

cypher

Copy code

MATCH (ja {name: "Ja Morant"}) DETACH DELETE ja;

**28. Query to delete a specific relationship between two nodes:**

cypher

Copy code

MATCH (lakers {name: "LA Lakers"})-[rel:PLAYS\_FOR]->(team:TEAM) DELETE rel;

**29. Query to delete all nodes from the graph model:**

cypher

Copy code

MATCH (n) DETACH DELETE n;

**GeoJSON to Represent Spatial Data JSON**

**Online /geojson.io**

json

Copy code

{

"type": "FeatureCollection",

"features": [

{

"type": "Feature",

"geometry": {

"type": "Point",

"coordinates": [102.0, 0.5]

},

"properties": {

"name": "Sample Point"

}

},

{

"type": "Feature",

"geometry": {

"type": "LineString",

"coordinates": [

[102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]

]

},

"properties": {

"name": "Sample Line"

}

}

]

}

Or

{

"type": "FeatureCollection",

"features": [

{

"type": "Feature",

"properties": {

"name": "Jammu & Kashmir",

"country": "India"

},

"geometry": {

"type": "Polygon",

"coordinates": [

[

[74.2402, 34.0688],

[74.4210, 33.9944],

[74.5605, 34.0429],

[74.7205, 33.8276],

[75.1580, 33.4412],

[75.4990, 33.3226],

[75.8057, 32.9948],

[76.2435, 33.0446],

[76.5827, 33.1944],

[76.7711, 33.4416],

[77.1737, 33.7001],

[77.8374, 34.0836],

[78.3792, 34.3219],

[78.9123, 34.5276],

[78.9123, 34.8020],

[78.1860, 35.4976],

[77.8374, 35.4976],

[76.7532, 35.3565],

[76.0494, 35.0454],

[75.5011, 34.6448],

[75.2584, 34.5563],

[74.9099, 34.2146],

[74.6575, 34.0479],

[74.2402, 34.0688]

]

]

}

}

]

}

**3. Manipulating GeoJSON with Python**

1. Install the necessary Python libraries. In the terminal within VS Code, run:

bash

Copy code

pip install geojson geopandas folium

code

import geojson

# Load the GeoJSON data

with open('data.geojson') as f:

data = geojson.load(f)

# Filter and print specific features by property name

for feature in data['features']:

if feature['properties']['name'] == 'Sample Point':

print("Sample Point Geometry:", feature['geometry'])

1. Run the script in VS Code by pressing F5 or using the terminal:

bash

Copy code

python manipulate\_geojson.py

**4. Visualizing GeoJSON Data with Folium**

1. Create a new Python file named visualize\_geojson.py.
2. Add the following code to visualize the GeoJSON data using **Folium** and generate an HTML map file:

python

Copy code

import folium

# Create a Folium map object centered around the coordinates

m = folium.Map(location=[0.5, 102.0], zoom\_start=5)

# Load the GeoJSON data and add it to the map

folium.GeoJson('data.geojson').add\_to(m)

# Save the map to an HTML file

m.save('map.html')

1. Run this script in the terminal:

bash

Copy code

python visualize\_geojson.py

**Experiment 8: Creating Geospatial Data Models for Location Analyses**

**Objective:**

* Create and query geospatial data models for location-based analysis.

***Tasks:***

1. **Understanding Geospatial Data Models:**

* Learn about spatial data types like POINT, LINESTRING, and POLYGON.
* Study spatial databases like PostGIS and their capabilities.

1. **Designing a Geospatial Schema:**

* Design a schema to store geospatial data for location analysis.
* Include columns with spatial data types.

Example:

sql

Copy code

CREATE TABLE Locations (

    LocationID INT PRIMARY KEY,

    Name VARCHAR(50),

    Coordinates POINT

);

1. **Populating the Geospatial Model:**

* Insert geospatial data into the tables.
* Use functions to calculate distances, areas, and perform spatial joins.

Example:

sql

Copy code

INSERT INTO Locations (LocationID, Name, Coordinates)

VALUES (1, 'Central Park', ST\_GeomFromText('POINT(40.785091 -73.968285)'));

1. **Performing Spatial Queries:**

* Write queries to analyze geospatial data.
* Examples: Finding locations within a radius, calculating the distance between two points.

Example:

sql

Copy code

SELECT Name, ST\_Distance(Coordinates, ST\_GeomFromText('POINT(40.730610 -73.935242)')) AS Distance

FROM Locations

ORDER BY Distance;

**Expected Output:**

* Geospatial data model schema.
* Example queries with results and visualizations.

**Experiment 10: Creating and Managing Metadata for Given Dataset**

* **Creating a Metadata Template**
* You need to create a **metadata template** to describe your dataset. This will typically include field names, data types, descriptions, constraints, and example values. Here's how to create the metadata template.
* **Create a metadata template JSON file:**
* In VSCode, create a new file called metadata\_template.json.
* Use the following structure for the template:
* json
* Copy code
* {
* "fields": [
* {
* "Field Name": "CustomerID",
* "Data Type": "INT",
* "Description": "Unique identifier for each customer",
* "Constraints": "Primary Key, Not Null",
* "Example Values": "1001"
* },
* {
* "Field Name": "CustomerName",
* "Data Type": "STR",
* "Description": "Name of the customer",
* "Constraints": "Not Null",
* "Example Values": "John Doe"
* },
* {
* "Field Name": "Email",
* "Data Type": "STR",
* "Description": "Customer's email address",
* "Constraints": "Unique, Not Null",
* "Example Values": "johndoe@example.com"
* }
* ]
* }
* **Step 3: Populating the Metadata Template**
* You can manually populate the metadata template for your entire dataset by adding fields as required. Add details like:
* Field name
* Data type (e.g., INT for integers, STR for strings)
* Description
* Constraints (e.g., Primary Key, Not Null)
* Example values
* **Step 4: Dataset Creation**
* Create a **sample dataset** in CSV format that you will validate. For example, create a file called data.csv in your VSCode project folder.
* Example data.csv:
* csv
* Copy code
* CustomerID,CustomerName,Email
* 1001,John Doe,johndoe@example.com
* 1002,Jane Smith,janesmith@example.com
* 1003,Bob Lee,boblee@example.com
* **Step 5: Writing the Python Validation Script**
* **Create a Python Script for Metadata Validation:**
* Create a new file in VSCode named validate\_metadata.py.
* Here's an example Python script that will validate your dataset against the metadata:
* python
* Copy code
* import pandas as pd
* import json
* # Load metadata template
* with open('metadata\_template.json') as f:
* metadata = json.load(f)
* # Load dataset
* df = pd.read\_csv('data.csv')
* # Function to validate data type
* def validate\_data\_type(column, data\_type):
* if data\_type == "INT":
* return all(isinstance(x, int) for x in df[column])
* elif data\_type == "STR":
* return all(isinstance(x, str) for x in df[column])
* return False
* # Validate dataset based on metadata
* for field in metadata['fields']:
* field\_name = field['Field Name']
* data\_type = field['Data Type']
* constraints = field['Constraints']
* # Check if column exists in dataset
* if field\_name not in df.columns:
* print(f"Error: {field\_name} is missing from dataset.")
* continue
* # Check for null values if 'Not Null' constraint is present
* if 'Not Null' in constraints and df[field\_name].isnull().any():
* print(f"Error: {field\_name} has null values!")
* # Validate data types
* if not validate\_data\_type(field\_name, data\_type):
* print(f"Error: {field\_name} has incorrect data types!")
* # Additional checks for constraints can be added here
* print("Metadata validation completed.")

**reverse engineer**

CREATE TABLE Passengers (

PassengerID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Gender VARCHAR(10),

DateOfBirth DATE,

ContactNumber VARCHAR(15),

Email VARCHAR(100)

);

-- Table for storing train details

CREATE TABLE Trains (

TrainID INT PRIMARY KEY AUTO\_INCREMENT,

TrainName VARCHAR(100) NOT NULL,

TrainType VARCHAR(50), -- e.g., Express, Local

Capacity INT NOT NULL

);

-- Table for storing station details

CREATE TABLE Stations (

StationID INT PRIMARY KEY AUTO\_INCREMENT,

StationName VARCHAR(100) NOT NULL,

Location VARCHAR(100) NOT NULL

);

-- Table for storing route details

CREATE TABLE Routes (

RouteID INT PRIMARY KEY AUTO\_INCREMENT,

TrainID INT NOT NULL,

DepartureStationID INT NOT NULL,

ArrivalStationID INT NOT NULL,

DepartureTime TIME NOT NULL,

ArrivalTime TIME NOT NULL,

Distance INT,

FOREIGN KEY (TrainID) REFERENCES Trains(TrainID),

FOREIGN KEY (DepartureStationID) REFERENCES Stations(StationID),

FOREIGN KEY (ArrivalStationID) REFERENCES Stations(StationID)

);

-- Table for storing booking information

CREATE TABLE Bookings (

BookingID INT PRIMARY KEY AUTO\_INCREMENT,

PassengerID INT NOT NULL,

RouteID INT NOT NULL,

BookingDate DATE NOT NULL,

JourneyDate DATE NOT NULL,

NumberOfSeats INT NOT NULL,

BookingStatus VARCHAR(20) DEFAULT 'Booked',

FOREIGN KEY (PassengerID) REFERENCES Passengers(PassengerID),

FOREIGN KEY (RouteID) REFERENCES Routes(RouteID)

);

-- Table for storing ticket payment information

CREATE TABLE Payments (

PaymentID INT PRIMARY KEY AUTO\_INCREMENT,

BookingID INT NOT NULL,

PaymentAmount DECIMAL(10, 2) NOT NULL,

PaymentDate DATE NOT NULL,

PaymentMethod VARCHAR(50), -- e.g., Credit Card, Debit Card, Net Banking

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID)

);

-- Table for storing seat availability for each route

CREATE TABLE SeatAvailability (

RouteID INT NOT NULL,

JourneyDate DATE NOT NULL,

AvailableSeats INT NOT NULL,

PRIMARY KEY (RouteID, JourneyDate),

FOREIGN KEY (RouteID) REFERENCES Routes(RouteID)

);

-- Table for storing train schedules

CREATE TABLE TrainSchedules (

ScheduleID INT PRIMARY KEY AUTO\_INCREMENT,

TrainID INT NOT NULL,

RouteID INT NOT NULL,

ScheduleDate DATE NOT NULL,

FOREIGN KEY (TrainID) REFERENCES Trains(TrainID),

FOREIGN KEY (RouteID) REFERENCES Routes(RouteID)

);