Neo4J –Query

- 1.Write a Neo4j Query that re3333

2. Write a Neo4j query to find the :Person node with name = 'Alice', add a new property city = 'Berlin', update her existing age property by adding 1, and assign the :Employee label to her.

Here is the Neo4j **Cypher** query to do all the specified operations on the `:Person` node with `name = 'Alice'`:

```
MATCH (p:Person {name: 'Alice'})
SET p.city = 'Berlin',
    p.age = p.age + 1
SET p :Employee;
```

Explanation:

- `MATCH (p:Person {name: 'Alice'})`: Finds the `:Person` node with `name = 'Alice'`.
- `SET p.city = 'Berlin'`: Adds a new property `city` with value `'Berlin'`.
- `p.age = p.age + 1`: Increments the existing `age` property by 1.
- `SET p :Employee`: Adds the `:Employee` label to the node (now has both `:Person` and `:Employee` labels).

3. Write a Neo4j to create a node representing a Person with properties: name, age, and city.

Create a `:Person` node with properties `name`, `age`, and `city`:

```
CREATE (:Person {name: 'John Doe', age: 30, city: 'New York'});
```

You can replace the values as needed.

4.Write a Neo4j query to create multiple nodes for Student with properties name, age, and department. Insert at least 3 students.

Create multiple `:Student` nodes (at least 3) with properties `name`, `age`, and `department`:

```
CREATE  (:Student {name: 'Alice', age: 21, department: 'Computer Science'}),
(:Student {name: 'Bob', age: 22, department: 'Mechanical Engineering'}),
  (:Student {name: 'Charlie', age: 20, department: 'Electrical Engineering'});
```

These `CREATE` statements will insert all three students in a single query.

5. Write a query to find all :Person nodes aged over 30 living in “Paris.”

Extend that query to only return those with at least one :FRIEND relationship to someone named “Alice.”

. Basic Query: Find all `:Person` nodes aged over 30 living in "Paris":

```
MATCH (p:Person)
WHERE p.age > 30 AND p.city = 'Paris'
RETURN p;
```

🔹 2. Extended Query: Return only those who have at least one `:FRIEND` relationship to a person named "Alice":

```
MATCH (p:Person)-[:FRIEND]->(friend:Person {name: 'Alice'})
WHERE p.age > 30 AND p.city = 'Paris'
RETURN p;
```

Explanation:

- `MATCH (p:Person)-[:FRIEND]->(friend:Person {name: 'Alice'})`: Finds `:Person` nodes who are friends with a person named "Alice".
- `WHERE p.age > 30 AND p.city = 'Paris'`: Filters those people by age and city.
- `RETURN p`: Returns the matching person nodes.

6. Explain how you can store a simple social-network graph (users and friendships) in a relational database.

- Sketch the tables and describe how you’d query “friends of friends.”

. Query: “Friends of Friends”

To find **friends of friends** (excluding direct friends and the user themself), say for user `1`:

```
SELECT DISTINCT foaf.id, foaf.name
FROM Friendships AS f1
JOIN Friendships AS f2 ON f1.friend_id = f2.user_id
JOIN Users AS foaf ON foaf.id = f2.friend_id
WHERE f1.user_id = 1
  AND foaf.id != 1
  AND foaf.id NOT IN (
    SELECT friend_id FROM Friendships WHERE user_id = 1
  );
```

Explanation:

- `f1.friend_id` is a direct friend of user `1`.
- `f2.friend_id` is a friend of that friend (friend of a friend).

- The `NOT IN` clause filters out direct friends and the user themself.