- **Neo4j Connectivity from the python jupyter :**

- How can we write a code to connect to neo4j desktop? how can we create node and relationship?
- To connect to a Neo4j database using Python and create nodes and relationships, you can use the Neo4j Python driver, which is officially supported by Neo4j. Here's how you can do it:
- **1. Install the Neo4j Python Driver:**
- You can install the Neo4j Python driver, known as neo4j or neo4j-driver, using pip:

Srno	Python code
1	pip install neo4j

- **2. Import the Required Modules:**
- In your Python script, import the necessary modules from the neo4j library:
-

Srno	Python code
2	from neo4j import GraphDatabase

- **3. Create a Connection to the Neo4j Database:**
- Define a class to create a connection to the Neo4j database:

Srno	Python code

3	class Neo4jConnector:
	  def __init__(self, uri, username, password):
	  self._driver = GraphDatabase.driver(uri, auth=(username, password))
	
	  def close(self):
	  self._driver.close()

4	Connect to the Neo4j database
	neo4j_uri = "bolt://localhost:7687" # Replace with your Neo4j URI
	neo4j_username = "neo4j" # Replace with your Neo4j username
	neo4j_password = "12345678" # Replace with your Neo4j password
	
	neo4j_driver = GraphDatabase.driver(neo4j_uri, auth=(neo4j_username, neo4j_password))

5	session = neo4j_driver.session() session

-
-

- **4. Create Nodes and Relationships:**

- You can define methods within the Neo4jConnector class to create nodes and relationships. Here's an example of how to create a node and a relationship:

Srno	Python code

4	class Neo4jConnector:
	  # ... (constructor and close method)
	
	  def create_person(self, name):
	  with self._driver.session() as session:
	  session.write_transaction(self._create_person, name)
	
	  @staticmethod
	  def _create_person(tx, name):
	  query = (
	  "CREATE (p:Person {name: $name})"
	  )
	  tx.run(query, name=name)
	
	  def create_knows_relationship(self, person1, person2):
	  with self._driver.session() as session:
	  session.write_transaction(self._create_knows_relationship, person1, person2)
	
	  @staticmethod
	  def _create_knows_relationship(tx, person1, person2):
	  query = (
	  "MATCH (p1:Person), (p2:Person) "
	  "WHERE p1.name = $person1 AND p2.name = $person2 "
	  "CREATE (p1)-[:KNOWS]->(p2)"
	  )
	  tx.run(query, person1=person1, person2=person2)

- In this example, create_person creates a node labeled "Person" with a "name" property, and create_knows_relationship creates a "KNOWS" relationship between two Person nodes.

- **5. Usage:**

- Now, you can create an instance of Neo4jConnector and use its methods to interact with the Neo4j database:

-

Srno	Python code

5	neo4j_conn = Neo4jConnector("bolt://localhost:7687", "your_username", "your_password")
	
	# Create nodes and relationships
	neo4j_conn.create_person("Alice")
	neo4j_conn.create_person("Bob")
	neo4j_conn.create_knows_relationship("Alice", "Bob")
	
	# Close the connection when done
	neo4j_conn.close()

- Make sure to replace "bolt://localhost:7687", "your_username", and "your_password" with your Neo4j database's connection details. This code demonstrates how to connect to Neo4j, create nodes, and establish relationships using the Neo4j Python driver.

CYPHER QUERIES STEP BY STEP FOR THIS PROJECT

- ## 1. GRAPHDB.csv Loading in the Neo4j Desktop:
- **If we want to Load CSV file , so we need choose any step**
-

Srno	Cypher Query
1	• LOAD CSV WITH HEADERS FROM 'file:///your_data.csv' AS row • MERGE (e:Employee {EmpNumber: row.EmpNumber, EmpName: row.EmpName}) • MERGE (d:Department {DeptName: row.DeptName}) • MERGE (e)-[:WORKS_IN]->(d)
2	• LOAD CSV WITH HEADERS FROM "file:///your_data.csv" AS row • CREATE (e:Employee { • EmpNumber: row.EmpNumber, • EmpDepartment: row.EmpDepartment, • EmpJobRole: row.EmpJobRole, • EmpPerformanceRate: toInteger(row.EmpPerformanceRate) • }) • CREATE (d:Department {Name: row.EmpDepartment})
3	• LOAD CSV WITH HEADERS FROM "file:///your_data.csv" as row with row where row.EmpNumber is not null • MERGE (n: EmpNumber { Name: row.EmpNumber}) • MERGE (m: EmpDepartment { Name: row.EmpDepartment}) • MERGE (n) - [: TO{ EmpDepartments: row.EmpDepartment}->(m)

- ## 2. Creating the Nodes to be connect or write in the Neo4j Desktop:

- **Creating the Nodes**
- **To be check blank nodes**
- **MATCH (blankNode)**
- **WHERE blankNode.Name IS NULL AND size(keys(blankNode)) = 0**
- **DETACH DELETE blankNode**

- **MATCH (node)**
- **WHERE NOT (node)--()**
- **RETURN node**

- **To be remove blank nodes**
- **MATCH (node)**
- **WHERE NOT (node)-[]-()**
- **DETACH DELETE node**

- **1. EmpNumbers nodes creation and status**
-

Srno	Cypher Query
1	MERGE (n:EmpNumbers{EmpNumbers: $EmpNumber, name: $EmpNumber})
2	MATCH (n:EmpNumbers) RETURN (n)
3	// If we want to show the EmpNumbers as a individually , the we can use this cypher. MATCH (n:EmpNumbers {EmpNumbers: 'E1001025'}) RETURN n LIMIT 25
4	MATCH (n:EmpNumbers) RETURN COUNT(n)
OUT	1200

- **2. EmpDepartments nodes creation and status**

Srno	Cypher Query
1	MERGE (n:EmpDepartments{EmpDepartments: $EmpDepartment, name: $EmpDepartment})
2	MATCH (n:EmpDepartments) RETURN (n)
3	MATCH (n:EmpDepartments) RETURN COUNT(n)
OUT	6

- **3. EmpJobRoles nodes creation and status**

Srno	Cypher Query
1	MERGE (n:EmpjobRoles{EmpjobRoles: $EmpJobRole, name: $EmpJobRole})
2	MATCH (n:EmpjobRoles) RETURN (n)
3	MATCH (n:EmpjobRoles) RETURN COUNT(n)
OUT	19

- **4. PerformanceRating nodes creation and status**

Srno	Cypher Query
1	MERGE (n:PerformanceRatings{PerformanceRatings: $PerformanceRating, name: $PerformanceRating})
2	MATCH (n:PerformanceRatings) RETURN (n)
3	MATCH (n:PerformanceRatings) RETURN COUNT (n)
Out	3

- ## 3. Creating the Relationships to be connect or write in the Neo4j Desktop:
- **Creating the Relationships between the nodes :**
- **1. EmpNumbers to EmpDepartments nodes of Creating Relationships and status**

Srno	Cypher Query
1	MERGE (A:EmpNumbers) MERGE (B:EmpDepartments) MERGE (A)-[r:WORKS_IN]->(B)
2	MATCH (A:EmpNumbers)-[r:WORKS_IN]->(B:EmpDepartments) RETURN A, r, B
3	// To be displaying individual EmpDepartment wise as per like it MATCH (A:EmpNumbers)-[r:WORKS_IN]->(B:EmpDepartments) WHERE B.EmpDepartments = 'Data Science' RETURN A, r, B

- **2. EmpNumbers to EmpJobRole nodes of Creating Relationships and status**

Srno	Cypher Query
1	MATCH(A:EmpNumbers) MATCH(B:EmpjobRoles) MERGE(A)-[r:HAS_ROLE]->(B)
2	MATCH (A:EmpNumbers)-[r:HAS_ROLE]->(B:EmpJobRole) RETURN A, r, B

- **3. EmpNumbers to PerformanceRating nodes of Creating Relationships and status**
-

Srno	Cypher Query
1	MATCH(A:EmpNumbers) MATCH(B:PerformanceRatings) MERGE(A)-[r:HIGH_PERFORMANCE]->(B)
2	MATCH (A:EmpNumbers)-[r:HIGH_PERFORMANCE]->(B:PerformanceRating) RETURN A, r, B
3	MATCH (A:EmpNumbers)-[r:HIGH_PERFORMANCE]->(B:PerformanceRatings) WHERE B.PerformanceRatings = 4 // Assuming 4 is the high performance rating RETURN A, r, B
4	MATCH (A:EmpNumbers)-[r:HIGH_PERFORMANCE]->(B:PerformanceRatings) WHERE B.PerformanceRatings = 4 WITH A, B RETURN B.PerformanceRatings AS Rating, COLLECT(A) AS Employees,

	COUNT(A) AS Count
5	MATCH (A:EmpNumbers)-[r:HIGH_PERFORMANCE]->(B:PerformanceRatings) WHERE B.PerformanceRatings = 4 WITH A, B RETURN B.PerformanceRatings AS Rating, COLLECT(A) AS Employees, COUNT(B) AS Count

- ## 4. Aggregation of the Nodes & Relationships

Srno	Cypher Query
1	MATCH (A:EmpNumbers)-[r:WORKS_IN]->(B:EmpDepartments) RETURN A.EmpNumbers AS Emp_Numbers, TYPE(r) AS relationship_type, B.EmpDepartments AS related_EmpDepartments
out	// Its displaying with all EmpNumbers to EmpDepartments with all departments Emp_Numbers relationship_type related_EmpDepartments "E1001264" "WORKS_IN" "Sales"
2	// To be displaying individual Deparmentwise MATCH (A:EmpNumbers)-[r:WORKS_IN]->(B:EmpDepartments) WHERE B.EmpDepartments = 'Data Science' RETURN A.EmpNumbers AS Emp_Numbers, TYPE(r) AS relationship_type, B.EmpDepartments AS related_EmpDepartments
out	Emp_Numbers relationship_type related_EmpDepartments "E100313" "WORKS_IN" "Data Science"