

Task

CRUD operations in graph databases

AIM: To perform CRUD operations like creating, inserting, querying, finding, deleting, operations on graph spaces with the steps to get started with Neo4j's Aura Graph

Data base:

Step 1: Copy and paste the following link into your web browser.

Step 2: Click on "Start Free!"

Step 3: Choose the option to continue "with google!"

Step 4: Click the "open in browser" button.

Step 5: After clicking "open", a text file will be automatically downloaded. This file contains your user ID and password details.

Step 6: Copy the password from the downloaded text file and paste it where required.

Step 7: Close the "not started with Neo4j" with beginner guides if it's open.

Step 8: You're now ready to begin practicing with the graph database.

How to create Node with properties
Properties are the key-value pairs using which a node stores data. Create a node with properties using the CREATE clause and need to specify these properties separated by commas within the flower braces "{}".
Syntax:

```
CREATE (node:{label{key}}: value1:key.1; value2:--  
key.2) return node
```

To verify the creation of the node, type and execute the following query in the dollar prompt
syntax:

```
match (n) return n
```

Creating Relations

To create a relationship using the CREATE clause and specify relationship within the square braces "[]" depending on the direction of the relationship, it is placed between hyphen "-" and arrow as shown in the following syntax.

`CREATE (node-1)-[:Relationship type]->(node-2)`

Syntax:

`CREATE (a:LabelOfNode1), (b:LabelOfNode2)`

`WHERE a.name = "nameofnode1" AND b.name = "nameofnode2"`

`CREATE (a)-[:RelationshipType]->(b) RETURN a, b`

Deleting a particular Node

To delete a particular node and need to specify the details of the node in the place of "n" in the above query.

Syntax:

`DELETE {node:label & properties} OR — .3) DELETE nodeLabel, label`

Create a graph database for student course registration created student and dept node and itself value of properties.

Create a cricket Board Node:

`CREATE (cb: CricketBoard {BoardID: 1, Name: 'Chennai Cricket Board', Address: 'Chennai', Phone: 9988776655}) RETURN cb`

~~`CREATE (t1: team {teamID: 'C0B01', BoardID: 1, name: 'ABS express', coach: 'Dr. Ramkumar', captain: 'SAMPATH KUMAR'}) RETURN t1`~~

~~`CREATE (t1: team {teamID: 'C0B01', BoardID: 1, name: 'ABS express', coach: 'Dr. Ramkumar', captain: 'SAMPATH KUMAR'}) RETURN t1`~~

~~`CREATE (t1: team {teamID: 'C0B01', BoardID: 1, name: 'ABS express', coach: 'Dr. Ramkumar', captain: 'SAMPATH KUMAR'}) RETURN t1`~~

Create(t2: team {teamID: 'CCB002', BoardID: 'BF001',
Name: 'KVR Express', Coach: 'Mr. KARTHIKEYAN',
Captain: 'Y. JOHN') return t2

CREATE Player Nodes:

Create (P1: player {playerID: '1', teamID: 'CC
B01', Name: 'Raj', age: 23, DateofBirth: '29 -
JUN - 1995', playingRole: 'Bowler', email: 'balajiid
@gmail.com') return P1

Create (P2: player {playerID: '33', TeamID: '1
CC B01', Name: 'Anand', age: 23, DateofBirth: '09 -
JAN - 1999', Playing: 'Batsman', Email: 'balajiid
@gmail.com') return P2

Create (P3: player {playerID: '65', teamID: '
'CC B02', Name: 'Suresh', Age: 27, DateofBirth: '22
OCT - 1996', PlayingRole: 'Batsman', Email: 'suresh
@gmail.com') return P3

CREATE Relationship among cricket Board and
teams:

match Ccb: CricketBoard {BoardID: 'BF001'},
match Ccb: CricketBoard {BoardID: 'BF001'},
[t1: team {teamID: 'CC B01'}] create Ccb)-
[r1: has] \rightarrow (t1) return Ccb, r1, t1
match Ccb: CricketBoard {BoardID: 'BF001'},
[t2: team {teamID: 'CC B02'}] create Ccb)-[r2:
 \rightarrow (t2) return Ccb, r2, t2

CREATE RELATIONSHIP AMONG PLAYER
and teams:

match (P1: player {playerID: '1'}), (t1: team
{teamID: 'CC B01'}) create Cpl [r1: playFor] \rightarrow
(t1) return Cpl, r1, t1

match (P2: player {playerID: '33'}), (t1: team
{teamID: 'CC B01'}) create CP2-[r2: playFor] \rightarrow (t1)

match (p2: player {playerID: 165, p}) {
 team (teamID: 165, t) create (P2) -> (p, playerID: 165)
 return P2
}

match (p1: player {playerID: 75, p}) {
 team (teamID: 75, t) create (P1) -> (p1, playerID: 75)
 return P1
}

return (t, p1, p2, p1 == p2 ? "The two players are the same." : "The two players are different.")

Player p1 has playerID 75 and player p2 has playerID 165.
These two players are different.

The two players are the same.
Player p1 has playerID 75 and player p2 has playerID 75.
These two players are the same.

Player p1 has playerID 75 and player p2 has playerID 165.
These two players are different.

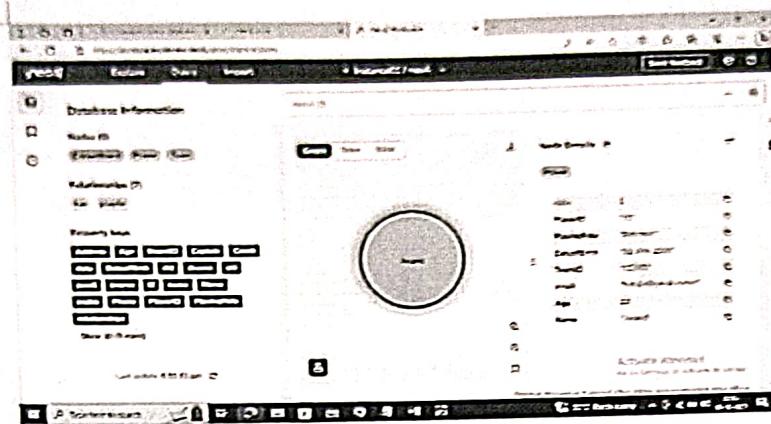
The screenshot shows the Neo4j browser interface. On the left, there's a sidebar with sections for 'Relationships (1)', 'Nodes (8)', and 'Property keys'. The main area displays a network graph with several nodes represented by circular icons. A specific node in the center has a red arrow pointing to it from the text below. To the right of the graph, there's a 'Results Overview' section with tabs for 'Nodes (8)', 'Relationships (1)', and 'Paths (0)'. Below the graph, a query bar contains the text 'match(p:Player{PlayerID:'33'}) return p'. The bottom status bar shows the system time as 'Last update 9:31:34 pm'.

OUTPUT:

This screenshot shows the Neo4j browser after running the query 'match(p:Player{PlayerID:'33'}) return p'. The results are displayed in a table titled 'Node Details' on the right side of the screen. The table has one row corresponding to the player with PlayerID 33. The columns show properties like 'PlayerID', 'TeamID', 'Name', and 'Custom Coach'. A red arrow points from the 'Name' column value 'DANNY COOPER' to the central node in the graph visualization above. The graph shows this node connected to other nodes, representing relationships. The sidebar on the left remains the same as in the first screenshot.

Retrieve particular player details:

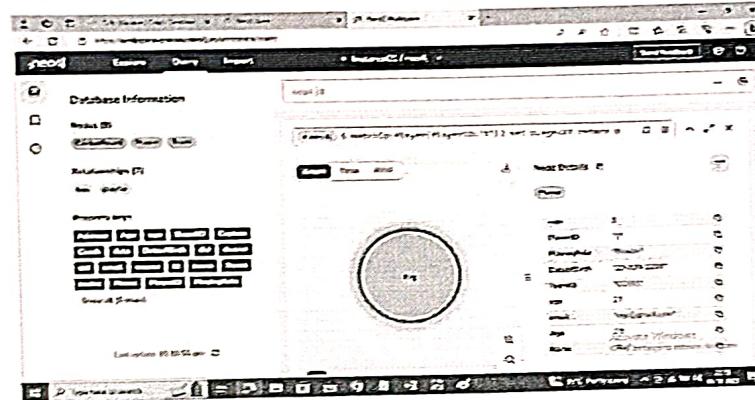
match(p:Player{PlayerID:'33'}) return p



Update particular player details:

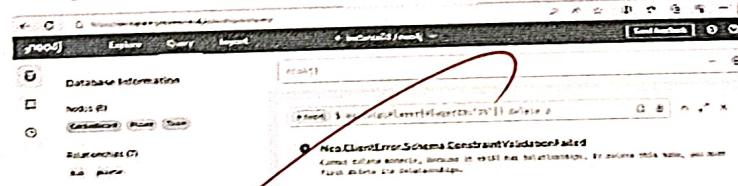
```
match(p:Player{PlayerID:'1'}) set p.age=27 return p
```

Output:



Delete particular player from the team:

```
match(p:Player{PlayerID:'33'}) delete p
```



Result:

Thus the CRUD operations like creating, inserting, querying, finding, deleting operations on graph spaces were executed successfully.

the graph database. It is a distributed system which stores data in a graph structure. Neo4j is a graph database management system. It is an open-source, native graph database management system. It is a highly performant, schema-free graph database. Neo4j is a distributed system which stores data in a graph structure.

Graph databases are used for many applications such as recommendation engines, fraud detection, social networks, and knowledge graphs. Neo4j is a distributed system which stores data in a graph structure.

Graph databases are used for many applications such as recommendation engines, fraud detection, social networks, and knowledge graphs. Neo4j is a distributed system which stores data in a graph structure.

Graph databases are used for many applications such as recommendation engines, fraud detection, social networks, and knowledge graphs. Neo4j is a distributed system which stores data in a graph structure.

VEL TECH - CSE	
EX NO.	
PERFORMANCE (5)	
RESULT AND ANALYSIS (3)	
VIVA VOCE (3)	
RECORD (4)	
TOTAL (15)	
IN WITH DATE	

~~Result~~ thus the CRUD operation like creating, inserting, updating, finding, deleting operations on graph stores were executed successfully.