Neo4j CQL - Introduction

Neo4j CQL

- Is a query language for Neo4j Graph Database.
- Is a declarative pattern-matching language.
- Follows SQL like syntax.
- Syntax is very simple and in human readable format.

Neo4j CQL Clauses

Following are the read clauses of Neo4j Cypher Query Language −

Sr.No	Read Clauses	Usage
1	MATCH	This clause is used to search the data with a specified pattern.
2	OPTIONAL MATCH	This is the same as match, the only difference being it can use nulls in case of missing parts of the pattern.
3	WHERE	This clause id is used to add contents to the CQL queries.
4	START	This clause is used to find the starting points through the legacy indexes.
5	LOAD CSV	This clause is used to import data from CSV files.

Following are the write clauses of Neo4j Cypher Query Language −

Sr.No	Write Clause	Usage
1	CREATE	This clause is used to create nodes, relationships, and properties.
2	MERGE	This clause verifies whether the specified pattern exists in the graph. If not, it creates the pattern.
3	SET	This clause is used to update labels on nodes, properties on nodes and relationships.
4	DELETE	This clause is used to delete nodes and relationships or paths etc. from the graph.
5	REMOVE	This clause is used to remove properties and elements from nodes and relationships.
6	FOREACH	This class is used to update the data within a list.

7	CREATE UNIQUE	Using the clauses CREATE and MATCH, you can get a unique pattern by matching the existing pattern and creating the missing one.
8	Importing CSV files with Cypher	Using Load CSV you can import data from .csv files.

Following are the general clauses of Neo4j Cypher Query Language −

Sr.No	General Clauses	Usage
1	RETURN	This clause is used to define what to include in the query result set.
2	ORDER BY	This clause is used to arrange the output of a query in order. It is used along with the clauses **RETURN** or **WITH**.
3	LIMIT	This clause is used to limit the rows in the result to a specific value.
4	SKIP	This clause is used to define from which row to start including the rows in the output.
5	WITH	This clause is used to chain the query parts together.
6	UNWIND	This clause is used to expand a list into a sequence of rows.
7	UNION	This clause is used to combine the result of multiple queries.
8	CALL	This clause is used to invoke a procedure deployed in the database.

Neo4j CQL Functions

Following are the frequently used Neo4j CQL Functions −

Sr.No	CQL Functions	Usage
1	String	They are used to work with String literals.
2	Aggregation	They are used to perform some aggregation operations on CQL Query results.
3	Relationship	They are used to get details of relationships such as startnode, endnode, etc.

Neo4j CQL Data Types

Sr.No	CQL Data Type	Usage
1	Boolean	It is used to represent Boolean literals: true, false.
2	byte	It is used to represent 8-bit integers.
3	short	It is used to represent 16-bit integers.
4	int	It is used to represent 32-bit integers.
5	long	It is used to represent 64-bit integers.
6	float	It is used to represent 32-bit floating-point numbers.
7	double	It is used to represent 64-bit floating-point numbers.
8	char	It is used to represent 16-bit characters.
9	String	It is used to represent Strings.

CQL Operators

Following are the list of operators supported by Neo4j Cypher Query language.

Sr.No	Type	Operators
1	Mathematical	+, -, *, /, %, ^
2	Comparison	+, <>, <, >, <=, >=
3	Boolean	AND, OR, XOR, NOT
4	String	+
5	List	+, IN, [X], [X…..Y]
6	Regular Expression	=-
7	String matching	STARTS WITH, ENDS WITH, CONSTRAINTS

Boolean Operators in Neo4j CQL

Neo4j supports the following Boolean operators to use in Neo4j CQL WHERE clause to support multiple conditions.

Sr.No	Boolean Operators	Description
1	AND	It is a Neo4j CQL keyword to support AND operation. It is like SQL AND operator.
2	OR	It is a Neo4j CQL keyword to support OR operation. It is like SQL AND operator.
3	NOT	It is a Neo4j CQL keyword to support NOT operation. It is like SQL AND operator.
4	XOR	It is a Neo4j CQL keyword to support XOR operation. It is like SQL AND operator.

Comparison Operators in Neo4j CQL

Neo4j supports the following Comparison operators to use in Neo4j CQL WHERE clause to support conditions.

Sr.No	Boolean Operators	Description
1	=	It is a Neo4j CQL "Equal To" operator.
2	< >	It is a Neo4j CQL "Not Equal To" operator.
3	<	It is a Neo4j CQL "Less Than" operator.
4	>	It is a Neo4j CQL "Greater Than" operator.
5	<=	It is a Neo4j CQL "Less Than Or Equal To" operator.
6	> =	It is a Neo4j CQL "Greater Than Or Equal To" operator.

Neo4j CQL - Creating Node

1. Creating a Single node

You can create a node in Neo4j by simply specifying the name of the node that is to be created along with the CREATE clause.

❖ Syntax

```
CREATE (node_name);
```

Note − Semicolon (;) is optional.

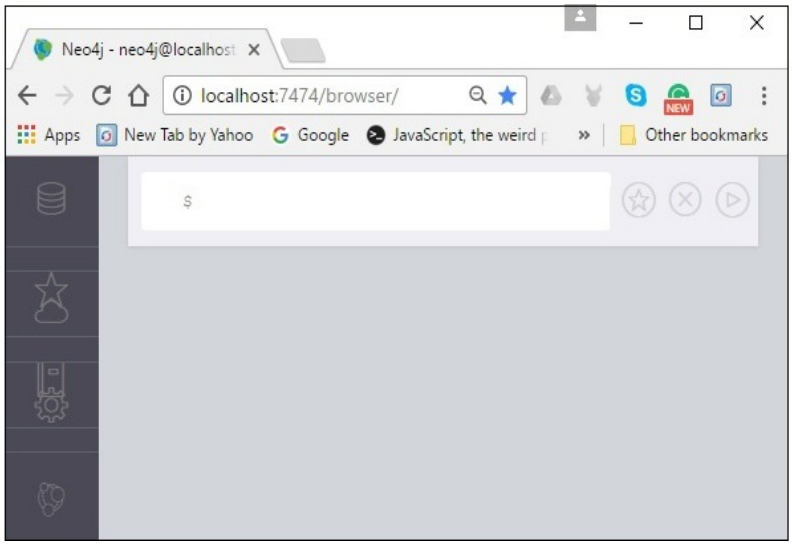
❖ Example

```
CREATE (sample)
```

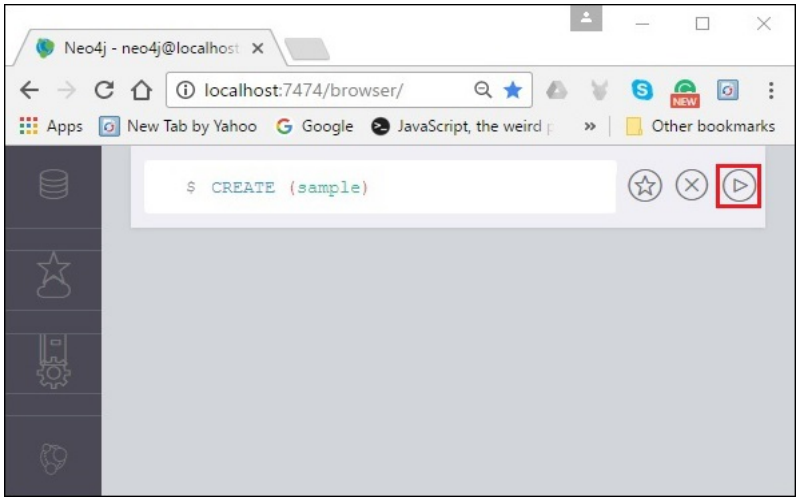
To execute the above query,carry out the following steps −

Step 1 − Open the Neo4j desktop App and start the Neo4j Server

Step 2 − Open your browser, copy paste the following URL in your address bar **http://localhost:7474/.** This will give you the built-in browser app of Neo4j with a dollar prompt

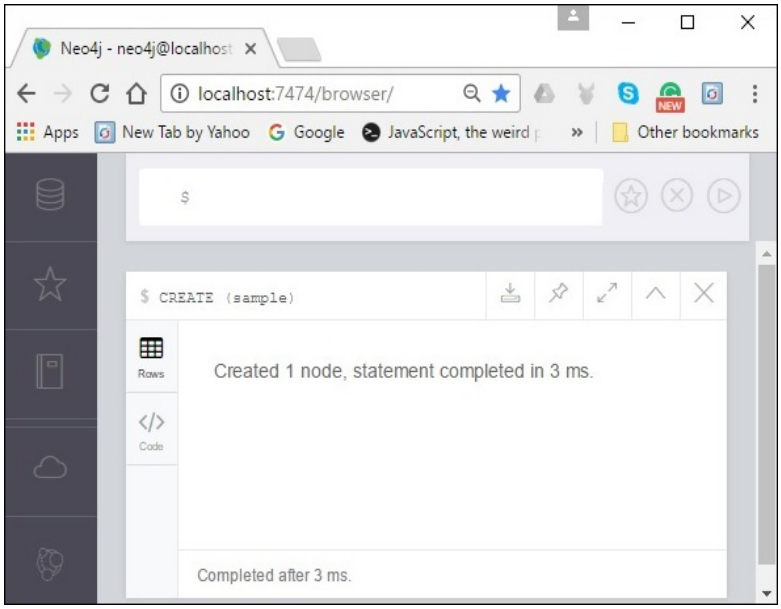


Step 3 − Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.



❖ Result

On executing, you will get the following result.

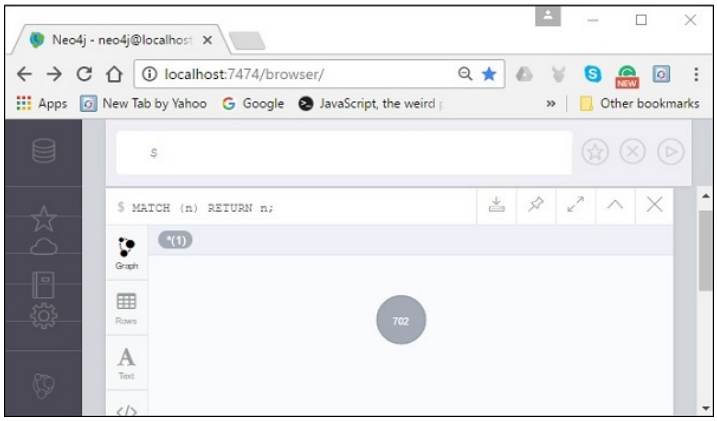


❖ Verification

To verify the creation of the node type, execute the following query in the dollar prompt.

MATCH (n) RETURN n

This query returns all the nodes in the database



2. Creating Multiple Nodes-

The create clause of Neo4j CQL is also used to create multiple nodes at the same time. To do so, you need to pass the names of the nodes to be created, separated by a comma.

❖ Syntax

Following is the syntax to create multiple nodes using the CREATE clause.

CREATE (node1),(node2)

❖ Example

Following is a sample Cypher Query which creates multiple nodes in Neo4j.

CREATE (sample1),(sample2)

❖ Verification

MATCH (n) RETURN n

3. Creating a Node with a Label

❖ Syntax

CREATE (node:label)

❖ Example

CREATE (Dhawan:player)

❖ **Verification**

```
MATCH (n) RETURN n
```

4. Creating a Node with Multiple Labels

You can also create multiple labels for a single node. You need to specify the labels for the node by separating them with a colon " : ".

❖ **Syntax**

```
CREATE (node:label1:label2:. . . . labeln)
```

❖ **Example**

```
CREATE (Dhawan:person:player)
```

❖ **Verification**

```
MATCH (n) RETURN n
```

5. Create Node with Properties

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

❖ **Syntax**

```
CREATE (node:label { key1: value, key2: value, . . . . . }) 
```

❖ **Example**

```
CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})
```

❖ **Verification**

```
MATCH (n) RETURN n
```

6. Returning the Created Node

Throughout the chapter, we used the **MATCH (n) RETURN n** query to view the created nodes. This query returns all the existing nodes in the database.

Instead of this, we can use the RETURN clause with CREATE to view the newly created node.

❖ **Syntax**

Following is the syntax to return a node in Neo4j.

```
CREATE (Node:Label{properties. . . .}) RETURN Node
```

❖ **Example**

```
CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"}) RETURN Dhawan
```

Neo4j CQL - Creating a Relationship

1. Creating Relationships

We can create a relationship using the CREATE clause. We will 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.

❖ **Syntax**

```
CREATE (node1)-[:RelationshipType]->(node2)
```

❖ **Example**

First of all, create two nodes Ind and Dhawan in the database, as shown below.

```
CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})
```

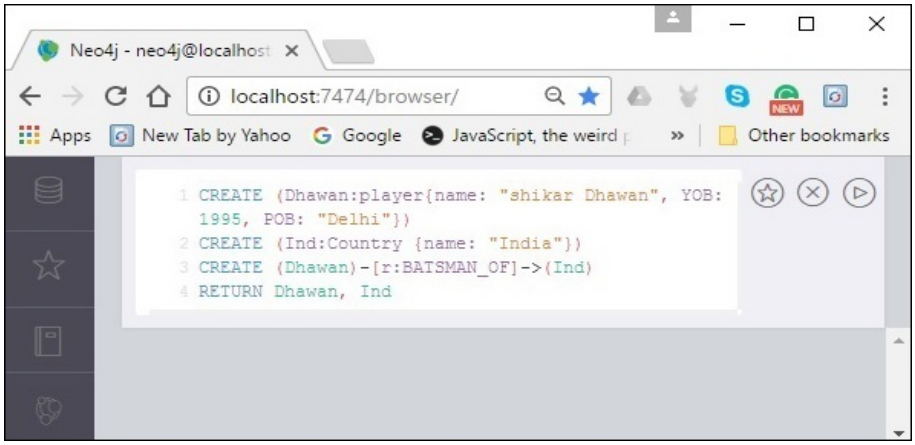
```
CREATE (Ind:Country {name: "India"})
```

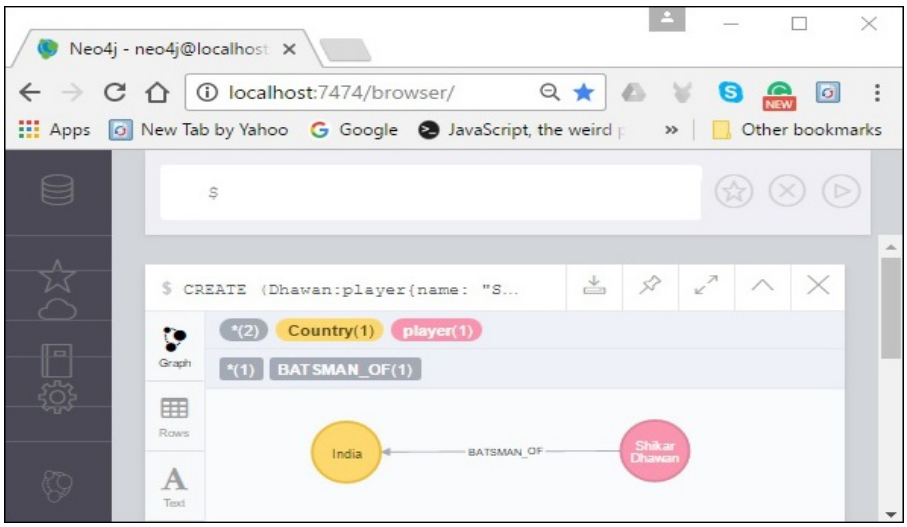
Now, create a relationship named **BATSMAN_OF** between these two nodes as −

```
CREATE (Dhawan)-[r:BATSMAN_OF]->(Ind)
```

Finally, return both the nodes to see the created relationship.

```
RETURN Dhawan, Ind
```





2. Creating a Relationship Between the Existing Nodes

You can also create a relationship between the existing nodes using the **MATCH** clause.

❖ Syntax

Following is the syntax to create a relationship using the MATCH clause.

```
MATCH (a:LabeofNode1), (b:LabeofNode2)
  WHERE a.name = "nameofnode1" AND b.name = " nameofnode2"
CREATE (a)-[: Relation]->(b)
RETURN a,b
```

❖ Example

```
MATCH (a:player), (b:Country) WHERE a.name = "Shikar Dhawan" AND b.name = "India"
CREATE (a)-[r: BATSMAN_OF]->(b)
RETURN a,b
```

3. Creating a Relationship with Label and Properties

You can create a relationship with label and properties using the CREATE clause.

❖ Syntax

Following is the syntax to create a relationship with label and properties using the CREATE clause.

```
CREATE (node1)-[label:Rel_Type {key1:value1, key2:value2, . . . n}]-> (node2)
```

❖ Example

Following is a sample Cypher Query which creates a relationship with label and properties.

```
MATCH (a:player), (b:Country) WHERE a.name = "Shikar Dhawan" AND b.name = "India"
CREATE (a)-[r:BATSMAN_OF {Matches:5, Avg:90.75}]->(b)
RETURN a,b
```

Creating a Complete Path

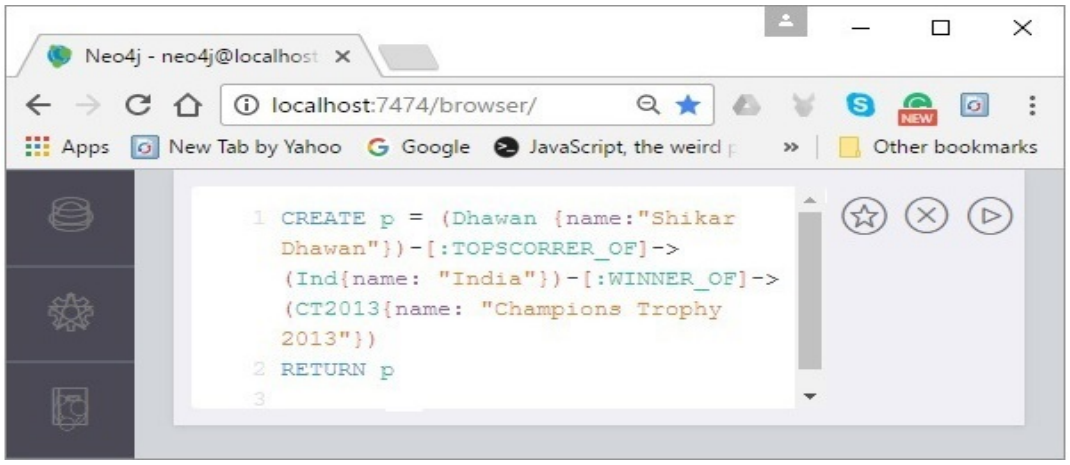
In Neo4j, a path is formed using continuous relationships. A path can be created using the create clause.

❖ Syntax

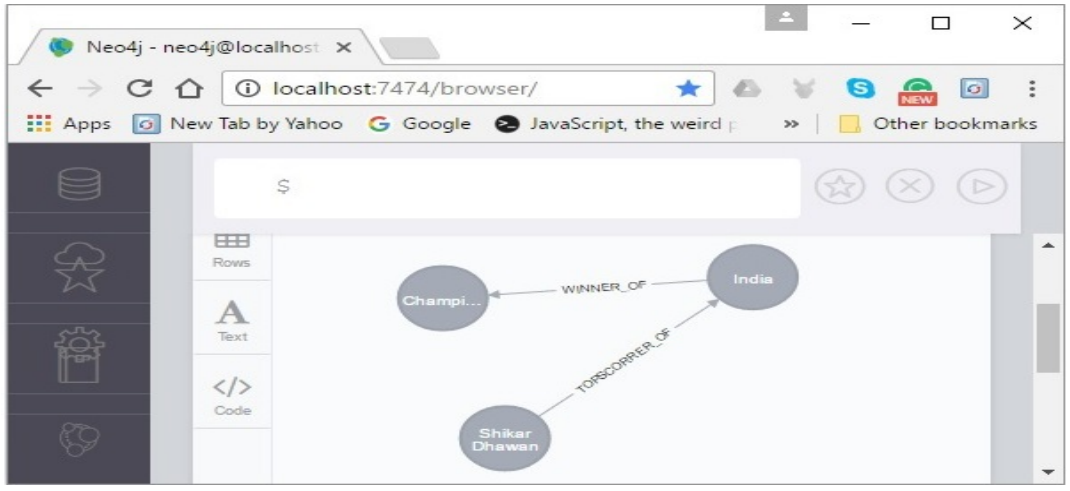
Following is the syntax to create a path in Neo4j using the CREATE clause.

```
CREATE p = (Node1 {properties})-[:Relationship_Type]->
  (Node2 {properties})[:Relationship_Type]->(Node3 {properties})
RETURN p
```

❖ Example



❖ Result



❖ Neo4j CQL Write Clauses

1. Merge
2. SET
3. DELETE

4. Remove
5. Foreach

Neo4j - Merge Command

MERGE command is a combination of CREATE command and MATCH command.
Neo4j CQL MERGE command searches for a given pattern in the graph. If it exists, then it returns the results. If it does NOT exist in the graph, then it creates a new node/relationship and returns the results.

❖ Syntax

Following is the syntax for the MERGE command.

```
MERGE (node: label {properties . . . . . })
```

///Before proceeding to the examples in this section, create two nodes in the database with labels Dhawan and Ind. Create a relationship of type “BATSMAN_OF” from Dhawan to Ind as shown below.

```
CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})
CREATE (Ind:Country {name: "India"})
CREATE (Dhawan)-[r:BATSMAN_OF]->(Ind)
```

1. Merging a Node with a Label

You can merge a node in the database based on the label using the MERGE clause. If you try to merge a node based on the label, then Neo4j verifies whether there exists any node with the given label. If not, the current node will be created.

❖ Syntax

Following is the syntax to merge a node based on a label.

```
MERGE (node:label) RETURN node
```

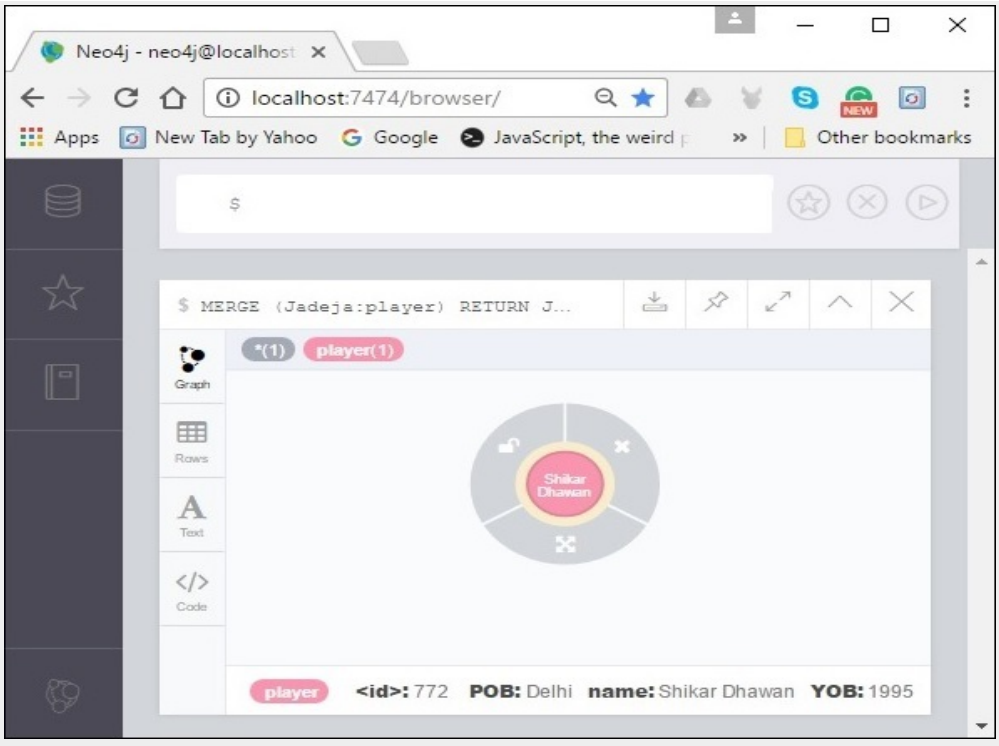
❖ Example

Following is a sample Cypher Query which merges a node into Neo4j (based on label). When you execute this query, Neo4j verifies whether there is any node with the label **player**. If not, it creates a node named “Jadeja” and returns it.
If, there exists any node with the given label, Neo4j returns them all.

```
MERGE (Jadeja:player) RETURN Jadeja
```

❖ Result

On executing, you will get the following result. Since you have already created a node named “Dhawan” with the label “player” in the database,



2. Merging a Node with Properties

You can also merge a node with a set of properties. If you do so, Neo4j searches for an equal match for the specified node, including the properties. If it doesn't find any, it creates one.

❖ Syntax

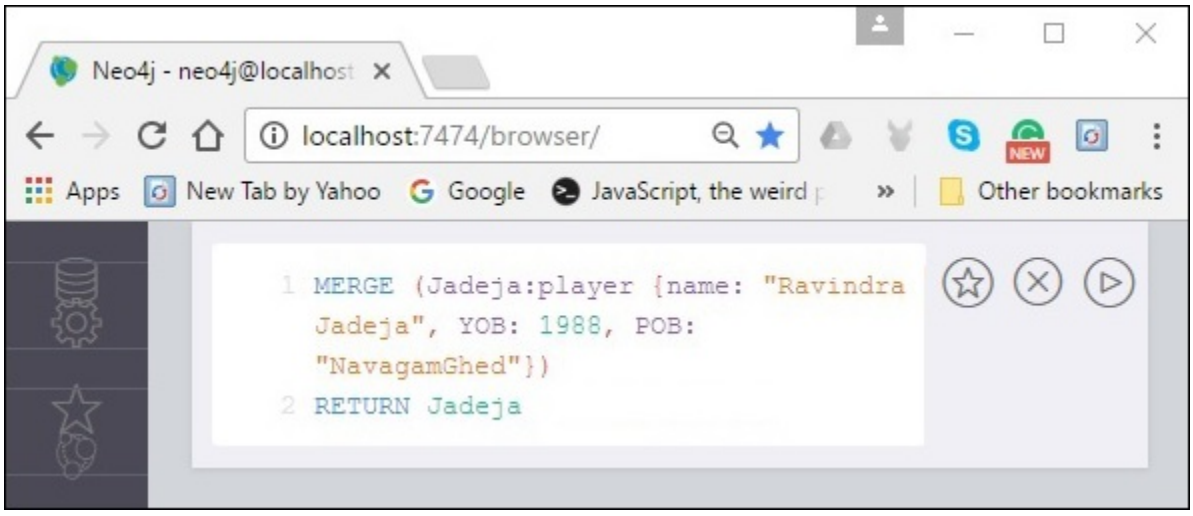
Following is the syntax to merge a node using properties.

```
MERGE (node:label {key1:value, key2:value, key3:value . . . . . })
```

❖ Example

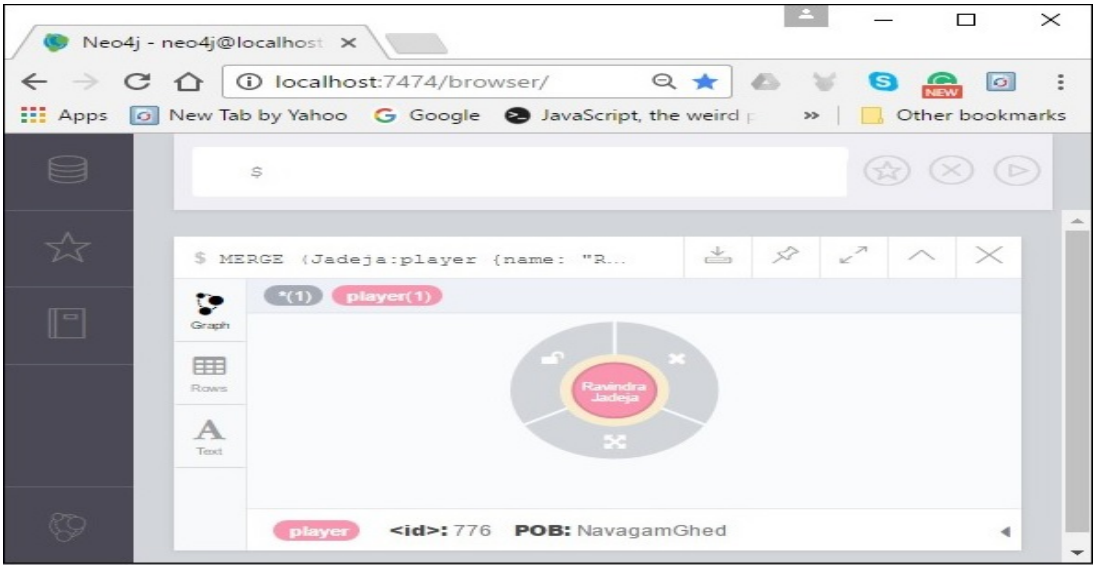
Following is a sample Cypher Query to merge a node using properties. This query tries to merge the node named "jadeja" using properties and label. Since there is no such node with the exact label and properties, Neo4j creates one.

```
MERGE (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB:
"NavagamGhed"})
RETURN Jadeja
```



❖ Result

On executing, you will get the following result. As discussed, since there are no nodes with the specified label and properties, it creates one



3. OnCreate and OnMatch

Whenever, we execute a merge query, a node is either matched or created. Using on create and on match, you can set properties for indicating whether the node is created or matched.

❖ Syntax

Following is the syntax of **OnCreate** and **OnMatch** clauses.

```
MERGE (node:label {properties . . . . . .})
ON CREATE SET property.isCreated ="true"
ON MATCH SET property.isFound ="true"
```

❖ Example

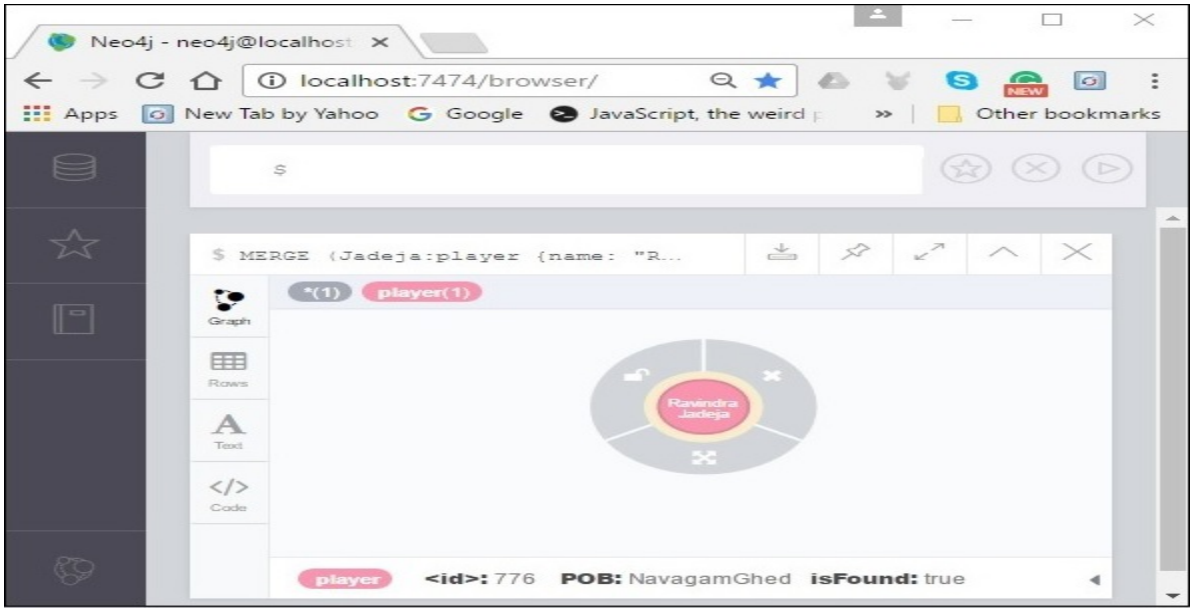
Following is a sample Cypher Query which demonstrates the usage of **OnCreate** and **OnMatch** clauses in Neo4j. If the specified node already exists in the database, then the node will be matched and the property with key-value pair isFound = "true" will be created in the node.

If the specified node doesn't exist in the database, then the node will be created, and within it a property with a key-value pair isCreated ="true" will be created.

```
MERGE (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB: "NavagamGhed"})
ON CREATE SET Jadeja.isCreated = "true"
ON MATCH SET Jadeja.isFound = "true"
RETURN Jadeja
```

❖ Result

On executing, you will get the following result. As discussed, since there is no node with the specified details, Neo4j created it along with the property **isFound**



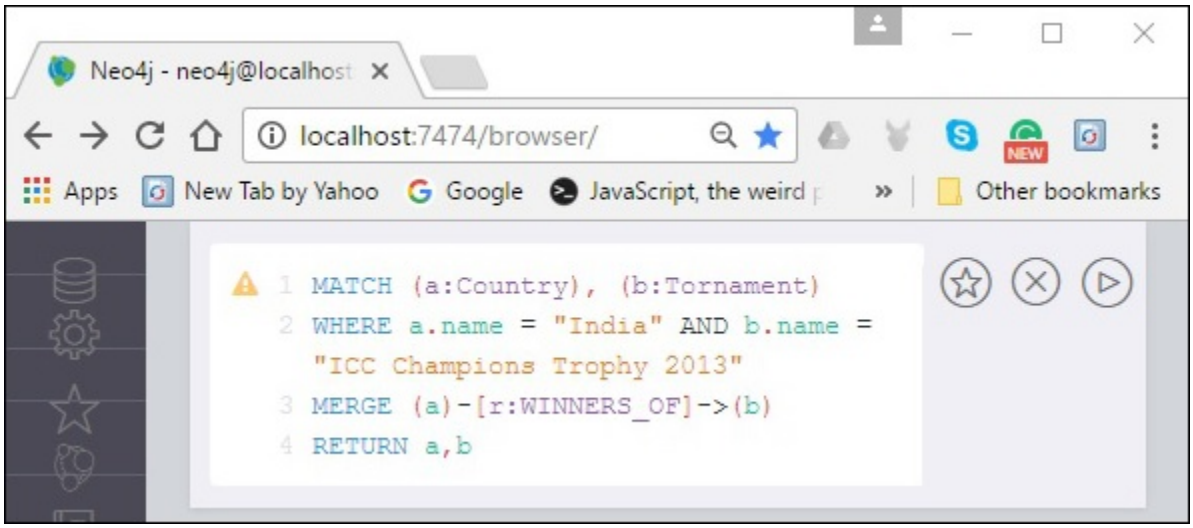
4. Merge a Relationship

Just like nodes, you can also merge the relationships using the MERGE clause.

❖ Example

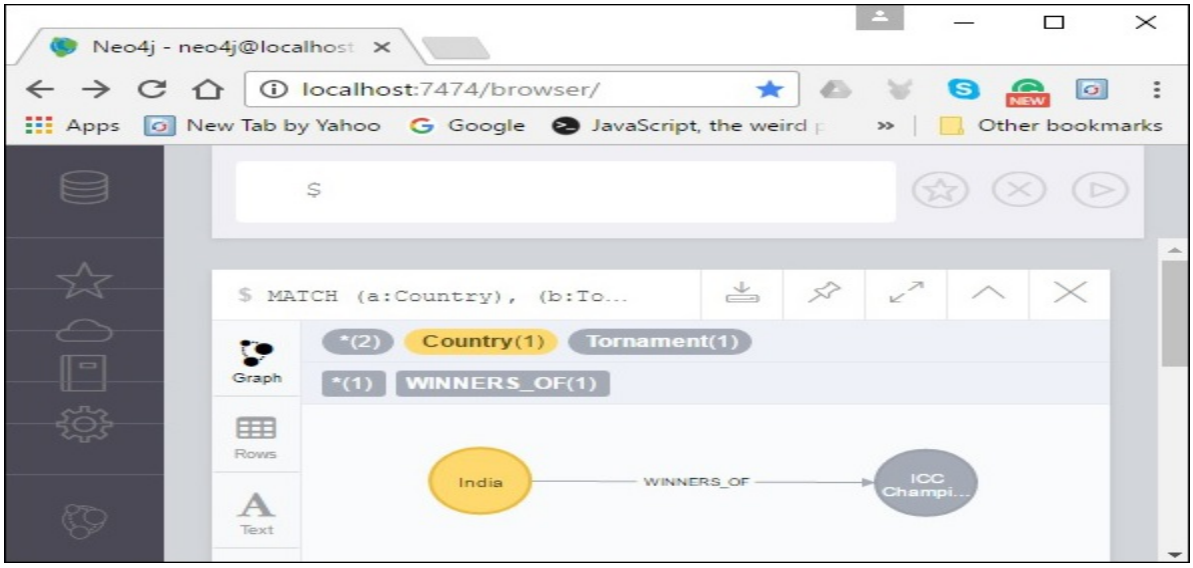
Following is a sample Cypher Query which merges a relationship using the MATCH clause in Neo4j. This query tries to merge a relationship named **WINNERS_OF** between the nodes "ind" (label: Country & name: India) and ICC13 (label: Tournament & name: ICC Champions Trophy 2013). Since such relation doesn't exist, Neo4j creates one.

```
MATCH (a:Country), (b:Tournament)
  WHERE a.name = "India" AND b.name = "ICC Champions Trophy 2013"
  MERGE (a)-[r:WINNERS_OF]->(b)
RETURN a, b
```



❖ Result

On executing, you will get the following result. Since the specified relation doesn't exist in the database, Neo4j creates one



Neo4j - Set Clause

Using Set clause, you can add new properties to an existing Node or Relationship, and also add or update existing Properties values.

1. Setting a Property

Using the SET clause, you can create a new property in a node.

Syntax

Following is the syntax for setting a property.

```
MATCH (node:label{properties . . . . . })

SET node.property = value

RETURN node
```

Example

Before proceeding with the example, first create a node named Dhawan as shown below.

```
CREATE (Dhawan:player{name: "shikar Dhawan", YOB: 1985, POB: "De
```

Following is a sample Cypher Query to create a property named "highestscore" with value "187".

```
MATCH (Dhawan:player{name: "shikar Dhawan", YOB: 1985, POB: "Delhi"})
```

```
SET Dhawan.highestscore = 187

RETURN Dhawan
```

2. Removing a Property

You can remove an existing property by passing **NULL** as value to it.

Syntax

Following is the syntax of removing a property from a node using the SET clause.

```
MATCH (node:label {properties})

SET node.property = NULL

RETURN node
```

Example

Before proceeding with the example, first create a node “jadeja” as shown below.

```
Create (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB:
"NavagamGhed"})
```

Following is a sample Cypher Query which removes the property named POB from this node using the SET clause as shown below.

```
MATCH (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB:
"NavagamGhed"})

SET Jadeja.POB = NULL

RETURN Jadeja
```

3. Setting Multiple Properties

In the same way, you can create multiple properties in a node using the Set clause. To do so, you need to specify these key value pairs with commas.

Syntax

Following is the syntax to create multiple properties in a node using the SET clause.

```
MATCH (node:label {properties})

SET node.property1 = value, node.property2 = value

RETURN node
```

Example

Following is a sample Cypher Query which creates multiple properties in a node using the SET clause in Neo4j.

```
MATCH (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988})

SET Jadeja.POB: "NavagamGhed", Jadeja.HS = "90"

RETURN Jadeja
```

4. Setting a Label on a Node

You can set a label to an existing node using the SET clause.

Syntax

Following is the syntax to set a label to an existing node.

```
MATCH (n {properties . . . . . })

SET n :label

RETURN n
```

Example

Before proceeding with the example, first create a node "Anderson" as shown below.

```
CREATE (Anderson {name: "James Anderson", YOB: 1982, POB: "Burnely"})
```

Following is a sample Cypher Query to set a label on a node using the SET clause. This query adds the label "player" to the node Anderson and returns it.

```
MATCH (Anderson {name: "James Anderson", YOB: 1982, POB: "Burnely"})

SET Anderson: player

RETURN Anderson
```

5. Setting Multiple Labels on a Node

You can set multiple labels to an existing node using the SET clause. Here you need to specify the labels by separating them with colons ":".

Syntax

Following is the syntax to set multiple labels to an existing node using the SET clause.

```
MATCH (n {properties . . . . . })

SET n :label1:label2

RETURN n
```

Example

Before proceeding with the example, first create a node named "Ishant" as shown below.

```
CREATE (Ishant {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})
```

Following is a sample Cypher Query used to create multiple labels on a node using the SET clause.

```
MATCH (Ishant {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})

SET Ishant: player:person

RETURN Ishant
```

Neo4j - Delete Clause

You can delete nodes and relationships from a database using the DELETE clause.

1. Deleting All Nodes and Relationships

Following is the query to delete all the nodes and the relationships in the database using the DELETE clause.

Query

```
MATCH (n) DETACH DELETE n
```

2. Deleting a Particular Node

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

Syntax

Following is the syntax to delete a particular node from Neo4j using the DELETE clause.

```
MATCH (node:label {properties . . . . . })
DETACH DELETE node
```

Example

Before proceeding with the example, create a node "Ishant" in the Neo4j database as shown below.

```
CREATE (Ishant:player {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})
```

Following is a sample Cypher Query which deletes the above created node using the DELETE clause.

```
MATCH (Ishant:player {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})
DETACH DELETE Ishant
```

Neo4j - Remove Clause

The REMOVE clause is used to remove properties and labels from graph elements (Nodes or Relationships).

The main difference between Neo4j CQL DELETE and REMOVE commands is –

> DELETE operation is used to delete nodes and associated relationships.
> REMOVE operation is used to remove labels and properties.

Removing a Property

You can remove a property of a node using MATCH along with the REMOVE clause.

Syntax

Following is the syntax to remove a property of a node using the REMOVE clause.

```
MATCH (node:label{properties . . . . . })
REMOVE node.property
RETURN node
```

Example

Before proceeding with the example, create a node named **Dhoni** as shown below.

```
CREATE (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB:
```

Following is a sample Cypher Query to remove the above created node using the REMOVE clause.

```
MATCH (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB:
"Ranchi"})
REMOVE Dhoni.POB
RETURN Dhoni
```

Removing a Label From a Node

Similar to property, you can also remove a label from an existing node using the remove clause.

Syntax

Following is the syntax to remove a label from a node.

```
MATCH (node:label {properties . . . . . })
REMOVE node:label
```

```
RETURN node
```

Example

Following is a sample Cypher Query to remove a label from an existing node using the remove clause.

```
MATCH (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB:
"Ranchi"})
REMOVE Dhoni:player
RETURN Dhoni
```

Removing Multiple Labels

You can also remove multiple labels from an existing node.

Syntax

Following is the syntax to remove multiple labels from a node.

```
MATCH (node:label1:label2 {properties . . . . . })
REMOVE node:label1:label2
RETURN node
```

Example

Before proceeding with the example, create a node Ishant as shown below.

```
CREATE (Ishant:player:person {name: "Ishant Sharma", YOB: 1988, POB:
"Delhi"})
```

Following is a sample Cypher Query to remove multiple labels from a node.

```
MATCH (Ishant:player:person {name: "Ishant Sharma", YOB: 1988, POB:
"Delhi"})
REMOVE Ishant:player:person
RETURN Ishant
```

Neo4j - Foreach Clause

The **FOREACH** clause is used to update data within a list whether components of a path, or result of aggregation.

Syntax

Following is the syntax of the FOREACH clause.

```
MATCH p = (start node)-[*]->(end node)
WHERE start.node = "node_name" AND end.node = "node_name"
FOREACH (n IN nodes(p)| SET n.marked = TRUE)
```

Example

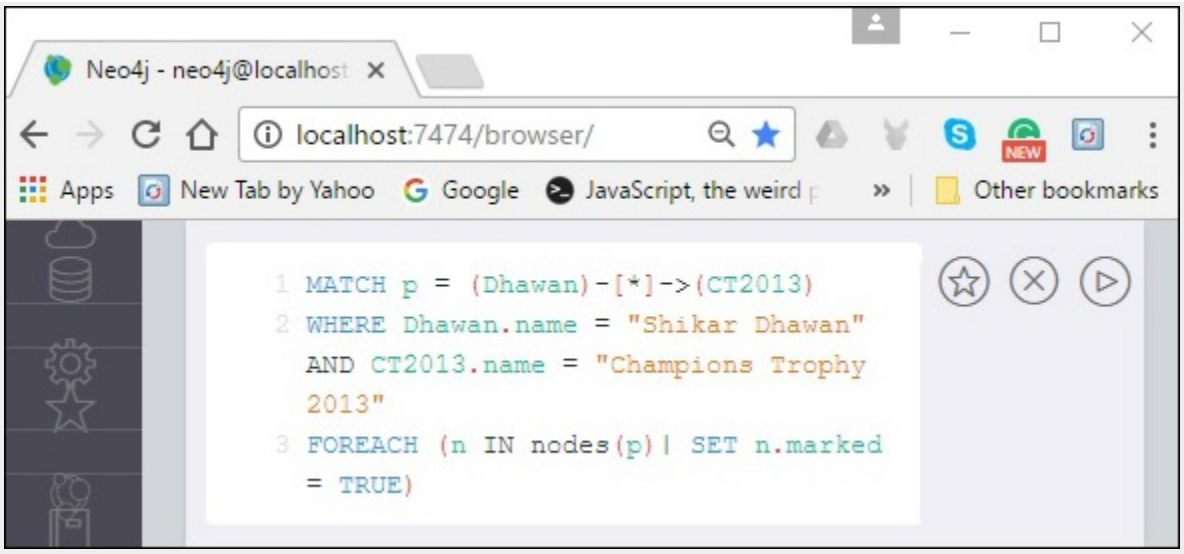
Before proceeding with the example, create a path **p** in Neo4j database as shown below.

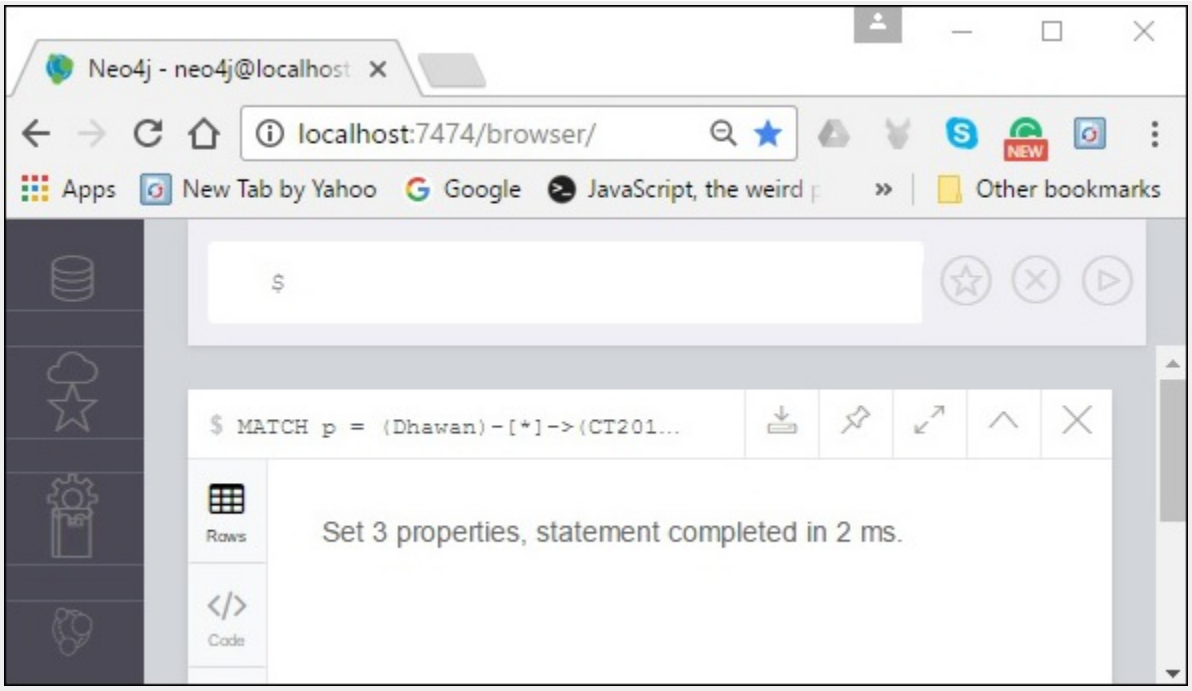
```
CREATE p = (Dhawan {name:"Shikar Dhawan"})-[:TOPSCORRER_OF]->(Ind{name:
   "India"})-[:WINNER_OF]->(CT2013{name: "Champions Trophy 2013"})
RETURN p
```

Following is a sample Cypher Query which adds a property to all the nodes along the path using the FOREACH clause.

```
MATCH p = (Dhawan)-[*]->(CT2013)
   WHERE Dhawan.name = "Shikar Dhawan" AND CT2013.name = "Champions
Trophy 2013"
FOREACH (n IN nodes(p)| SET n.marked = TRUE)
```

Result

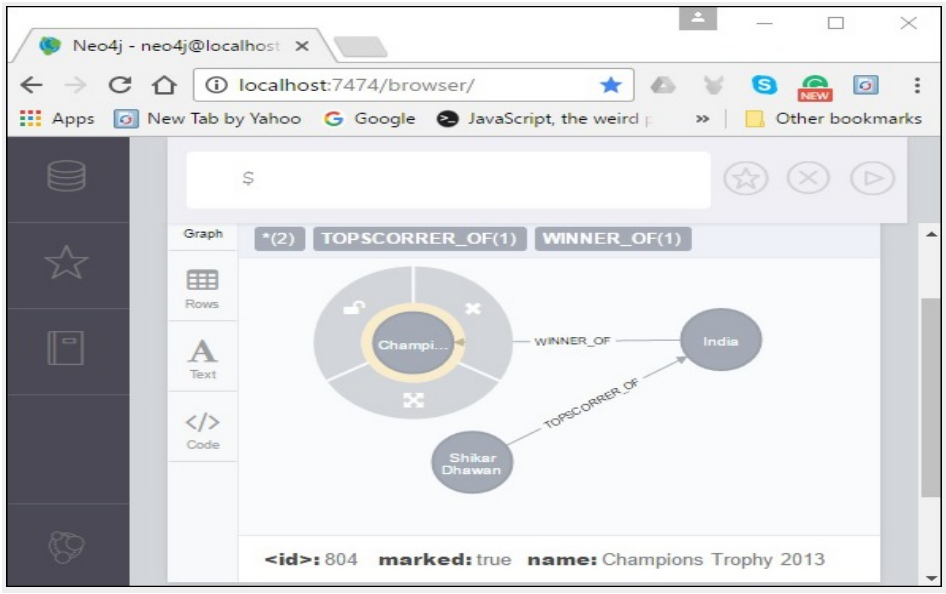




Verification

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

```
MATCH (n) RETURN n
```



❖ Neo4j CQL Write Clauses

1. Match
2. Optional match
3. Where
4. Count

Neo4j - Match Clause

1. Get All Nodes Using Match

Using the MATCH clause of Neo4j you can retrieve all nodes in the Neo4j database.

Syntax-
MATCH (n) RETURN n

2. Getting All Nodes Under a Specific Label

Using match clause, you can get all the nodes under a specific label.

Syntax

Following is the syntax to get all the nodes under a specific label.

```
MATCH (node:label)
RETURN node
```

Example

Following is a sample Cypher Query, which returns all the nodes in the database under the label **player**.

```
MATCH (n:player)
RETURN n
```

3. Match by Relationship

You can retrieve nodes based on relationship using the MATCH clause.

Syntax

Following is the syntax of retrieving nodes based on the relationship using the MATCH clause.

```
MATCH (node:label)<-[: Relationship]-(n)
RETURN n
```

Example

Following is a sample Cypher Query to retrieve nodes based on relationship using the MATCH clause.

```
MATCH (Ind:Country {name: "India", result: "Winners"})<-[:
TOP_SCORER_OF]-(n)
RETURN n.name
```

Neo4j - Optional Match Clause

The OPTIONAL MATCH clause is used to search for the pattern described in it, while using nulls for missing parts of the pattern.

OPTIONAL MATCH is similar to the match clause, the only difference being it returns null as a result of the missing parts of the pattern.

Syntax

Following is the syntax of the OPTIONAL MATCH with relationship.

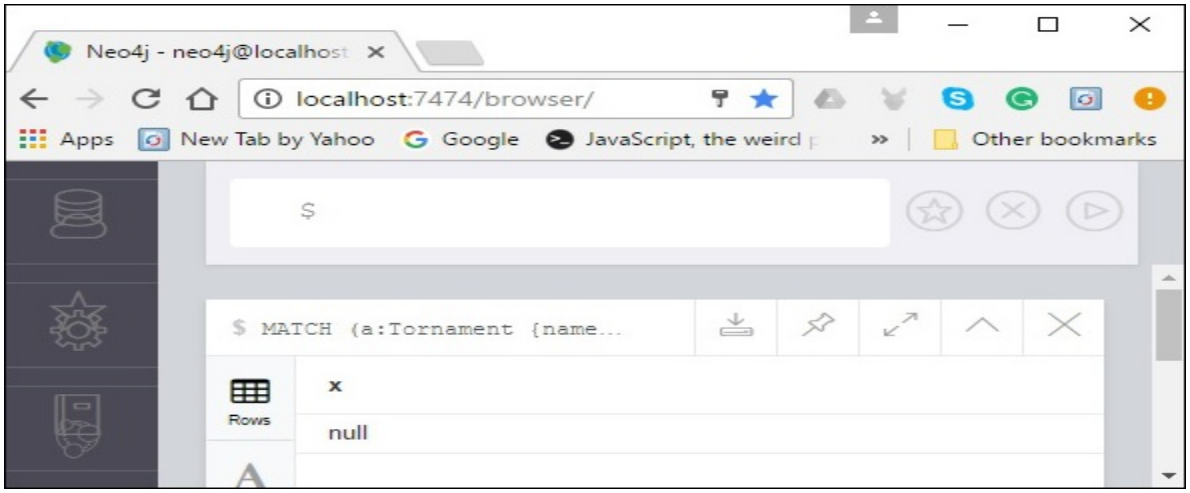
```
MATCH (node:label {properties. . . . . .})
OPTIONAL MATCH (node)-->(x)
RETURN x
```

Example

Following is a sample Cypher Query which tries to retrieve the relations from the node ICCT2013. Since there are no such nodes, it returns null.

```
MATCH (a:Tornament {name: "ICC Champions Trophy 2013"})
OPTIONAL MATCH (a)-->(x)
RETURN x
```

Result



Neo4j - Where Clause

1. Like SQL, Neo4j CQL has provided WHERE clause in CQL MATCH command to filter the results of a MATCH Query.

Syntax

Following is the syntax of the WHERE clause.

```
MATCH (label)
WHERE label.country = "property"
RETURN label
```

Example

Before proceeding with the example, create five nodes in the database as shown below.

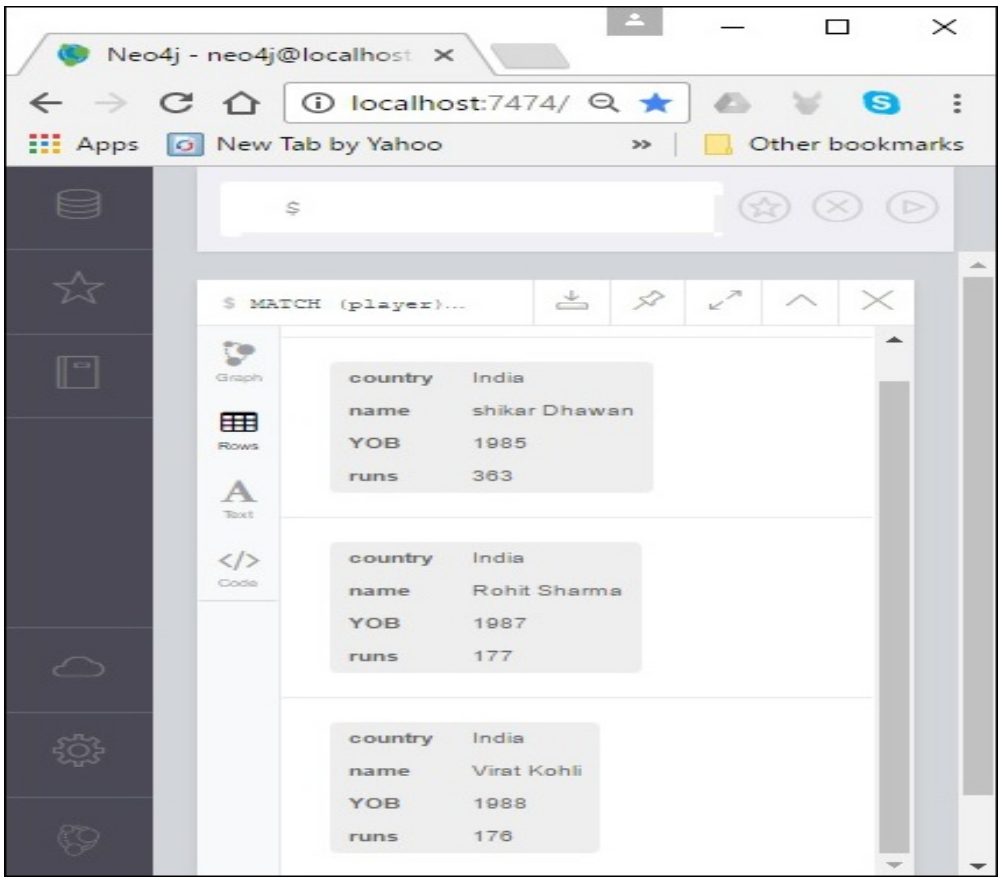
```
CREATE(Dhawan:player{name:"shikar Dhawan", YOB: 1985, runs:363, country:
"India"}

CREATE(Jonathan:player{name:"Jonathan Trott", YOB:1981, runs:229,
country:"South Africa"}

CREATE(Sangakkara:player{name:"Kumar Sangakkara", YOB:1977, runs:222,

   country:"Srilanka"})

CREATE(Rohit:player{name:"Rohit Sharma", YOB: 1987, runs:177,
country:"India"})

CREATE(Virat:player{name:"Virat Kohli", YOB: 1988, runs:176,
country:"India"})

CREATE(Ind:Country {name: "India", result: "Winners"})
```

Following is a sample Cypher Query which returns all the players (nodes) that belongs to the country India using WHERE clause.

```
MATCH (player)
WHERE player.country = "India"
```

```
RETURN player
```

Result



2. WHERE Clause with Multiple Conditions

You can also use the WHERE clause to verify multiple conditions.

Syntax

Following is the syntax to use WHERE clause in Neo4j with multiple conditions.

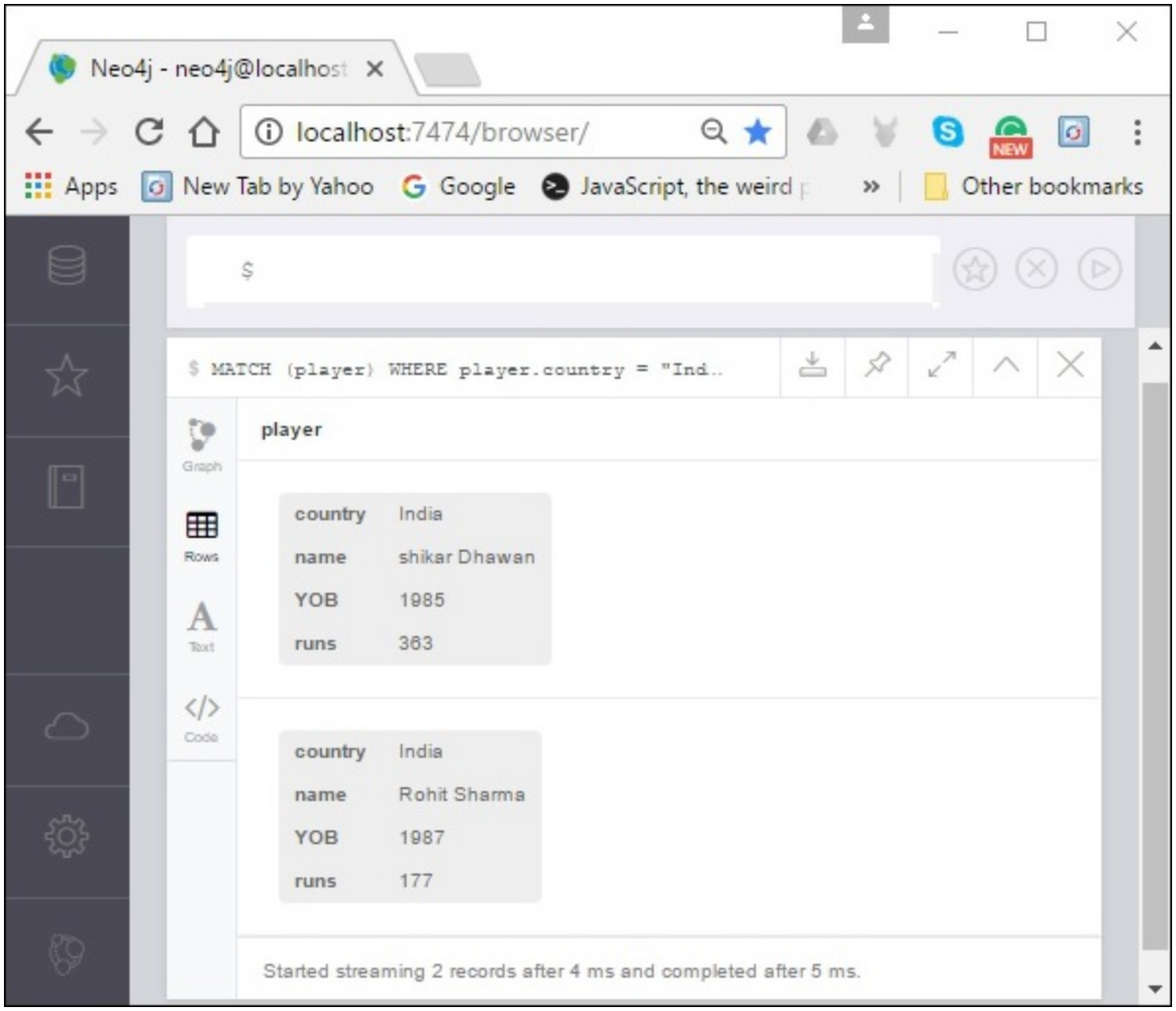
```
MATCH (emp:Employee)
WHERE emp.name = 'Abc' AND emp.name = 'Xyz'
RETURN emp
```

Example

Following is a sample Cypher Query which filters the nodes in the Neo4j database using two conditions.

```
MATCH (player)
WHERE player.country = "India" AND player.runs >=175
RETURN player
```

Result

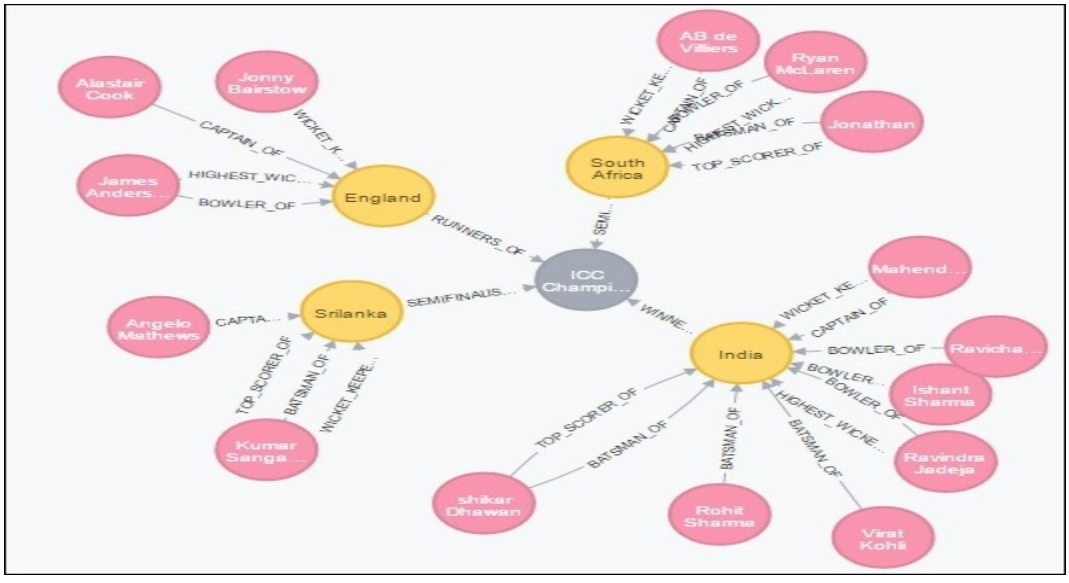


3. Using Relationship with Where Clause

You can also use Where clause to filter the nodes using the relationships.

Example

Assume we have the following graph in the database.

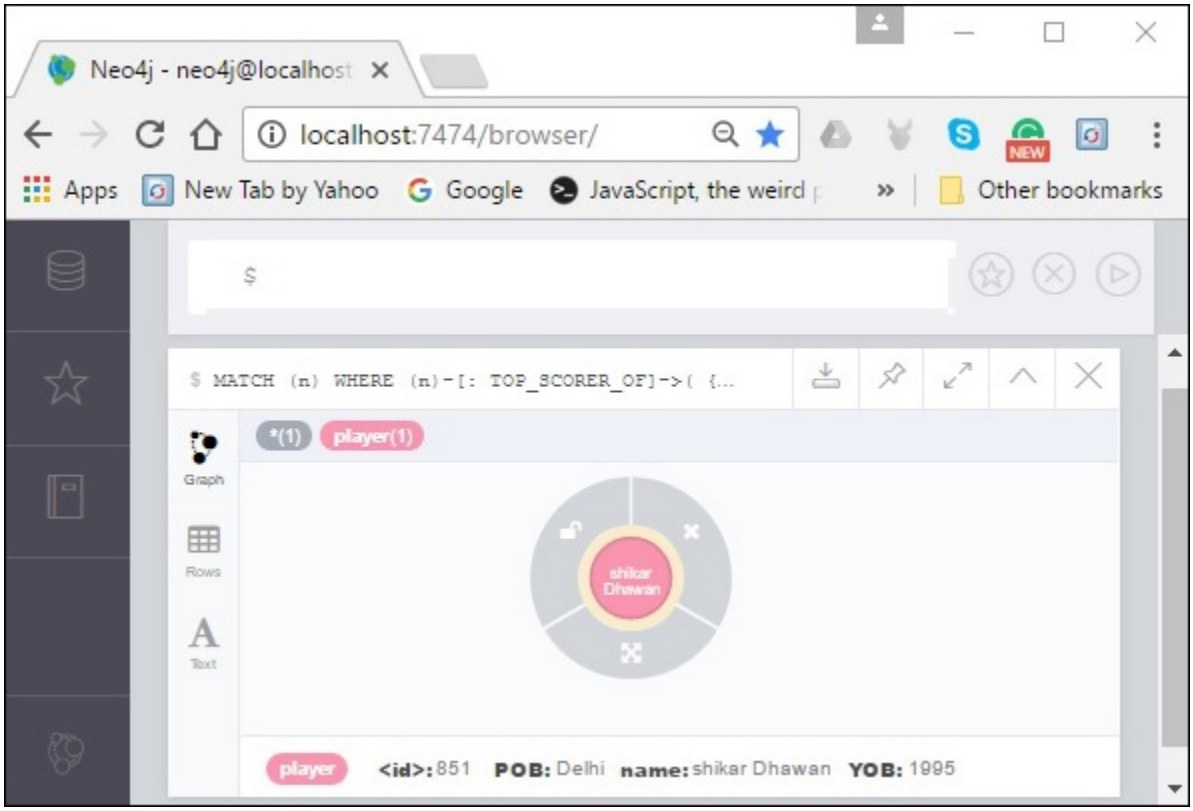


Following is a sample Cypher Query to retrieve the top scorer of India using WHERE clause as shown below.

```
MATCH (n)
WHERE (n)-[: TOP_SCORER_OF]->( {name: "India", result: "Winners"})
RETURN n
```

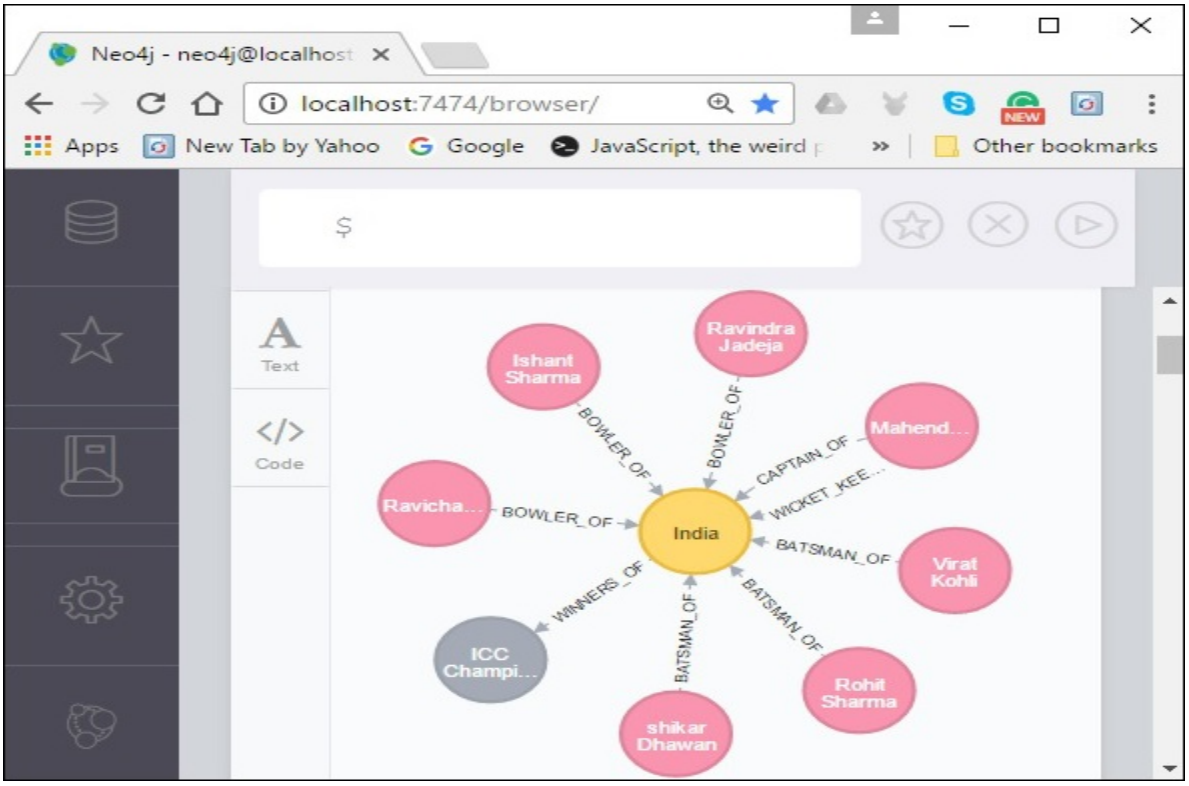
Result

On executing, you will get the following result. Here you can observe that Neo4j returned the node, which has the relation TOP_SCORER_OF to the country with the node having the name India.



Neo4j - Count Function

Assume we have created a graph in the database with the following details.



1. Count

The count() function is used to count the number of rows.

Syntax

Following is the syntax of the count function.

```
MATCH (n { name: 'A' })-->(x)
RETURN n, count(*)
```

Example

Following is a sample Cypher Query which demonstrates the usage of the **count()** function.

```
Match(n{name: "India", result: "Winners"})--(x)
RETURN n, count(*)
```

2. Group Count

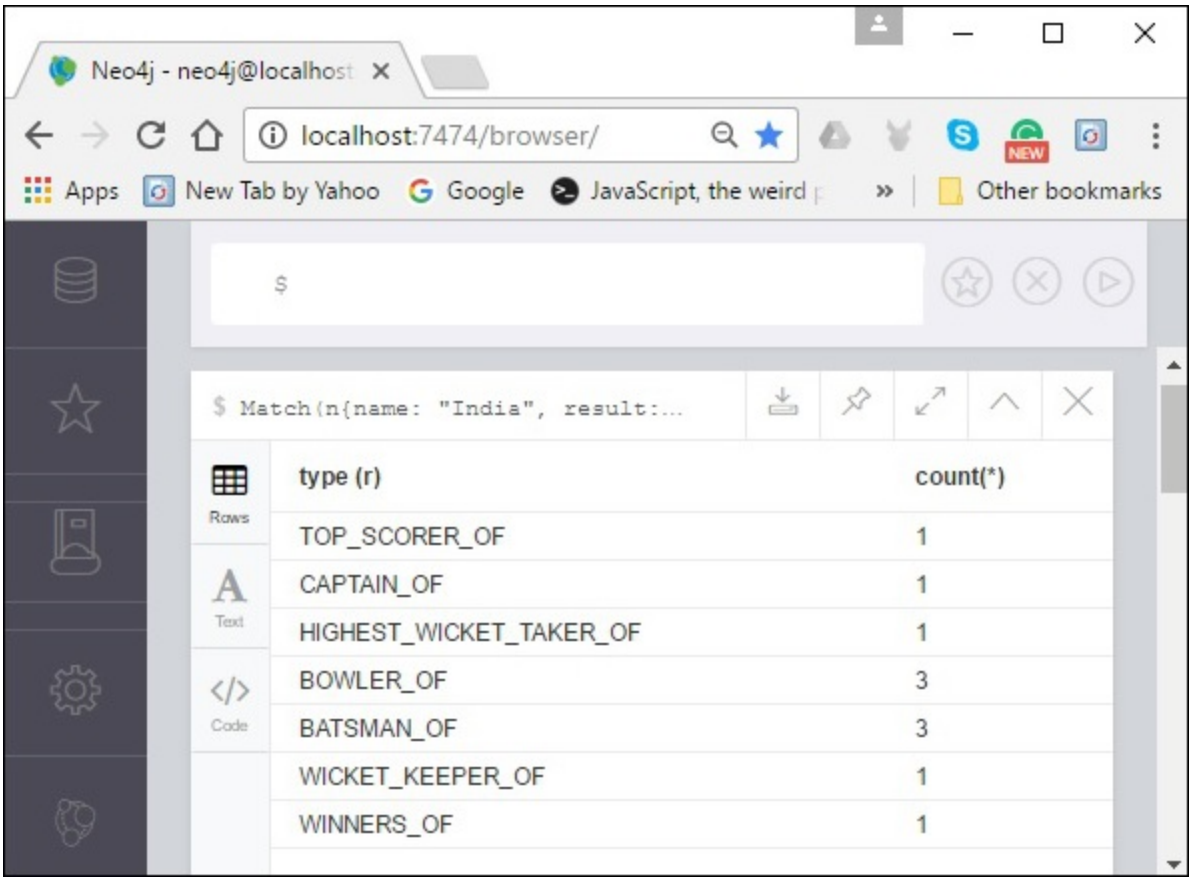
The **COUNT** clause is also used to count the groups of relationship types.

Example

Following is a sample Cypher Query which counts and returns the number of nodes participating in each relation.

```
Match(n{name: "India", result: "Winners"})-[r]-(x)
RETURN type (r), count(*)
```

Result



- **Neo4j CQL General Clauses-**
  1. Return
  2. Order-by
  3. Limit
  4. Skip
  5. With
  6. Unwind

Neo4j - Return Clause

The RETURN clause is used return nodes, relationships, and properties in Neo4j.

1. Returning Nodes

You can return a node using the RETURN clause.

Syntax

Following is a syntax to return nodes using the RETURN clause.

```
Create (node:label {properties})
RETURN node
```

Example

Before proceeding with the example, create 3 nodes and 2 relationships as shown below.

```
Create (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB:
"Ranchi"})
CREATE (Ind:Country {name: "India", result: "Winners"})
CREATE (CT2013:Tornament {name: "ICC Champions Trophy 2013"})
CREATE (Ind)-[r1:WINNERS_OF {NRR:0.938 ,pts:6}]->(CT2013)
CREATE(Dhoni)-[r2:CAPTAIN_OF]->(Ind)
```

Following is a sample Cypher Query which creates a node named Dhoni and returns it.

```
Create (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB:
"Ranchi"})
RETURN Dhoni
```

2. Returning Multiple Nodes

You can also return multiple nodes using the return clause.

Syntax

Following is the syntax to return multiple nodes using the return clause.

```
CREATE (Ind:Country {name: "India", result: "Winners"})
CREATE (CT2013:Tornament {name: "ICC Champions Trophy 2013"})
RETURN Ind, CT2013
```

3. Returning Relationships

You can also return relationships using the Return clause.

Syntax

Following is the syntax to return relationships using the RETURN clause.

```
CREATE (node1)-[Relationship:Relationship_type]->(node2)
RETURN Relationship
```

Example

Following is a sample Cypher Query which creates two relationships and returns them.

```
CREATE (Ind)-[r1:WINNERS_OF {NRR:0.938 ,pts:6}]->(CT2013)
CREATE(Dhoni)-[r2:CAPTAIN_OF]->(Ind)
RETURN r1, r2
```

4. Returning Properties

You can also return properties using the RETURN clause.

Syntax

Following is a syntax to return properties using the RETURN clause.

```
Match (node:label {properties . . . . . })
Return node.property
```

Example

Following is a sample Cypher Query to return the properties of a node.

```
Match (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB:
"Ranchi"})
Return Dhoni.name, Dhoni.POB
```

5. Returning All Elements

You can return all the elements in the Neo4j database using the RETURN clause.

Example

Following is an example Cypher Query to return all the elements in the database.

```
Match p = (n {name: "India", result: "Winners"})-[r]-(x)
RETURN *
```

6. Returning a Variable With a Column Alias

You can return a particular column with alias using RETURN clause in Neo4j.

Example

Following is a sample Cypher Query which returns the column POB as Place Of Birth.

```
Match (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB:
"Ranchi"})
Return Dhoni.POB as Place Of Birth
```

Neo4j - Order By Clause

1. You can arrange the result data in order using the ORDER BY clause.

Syntax

Following is the syntax of the ORDER BY clause.

```
MATCH (n)
RETURN n.property1, n.property2 . . . . . .
ORDER BY n.property
```

Example

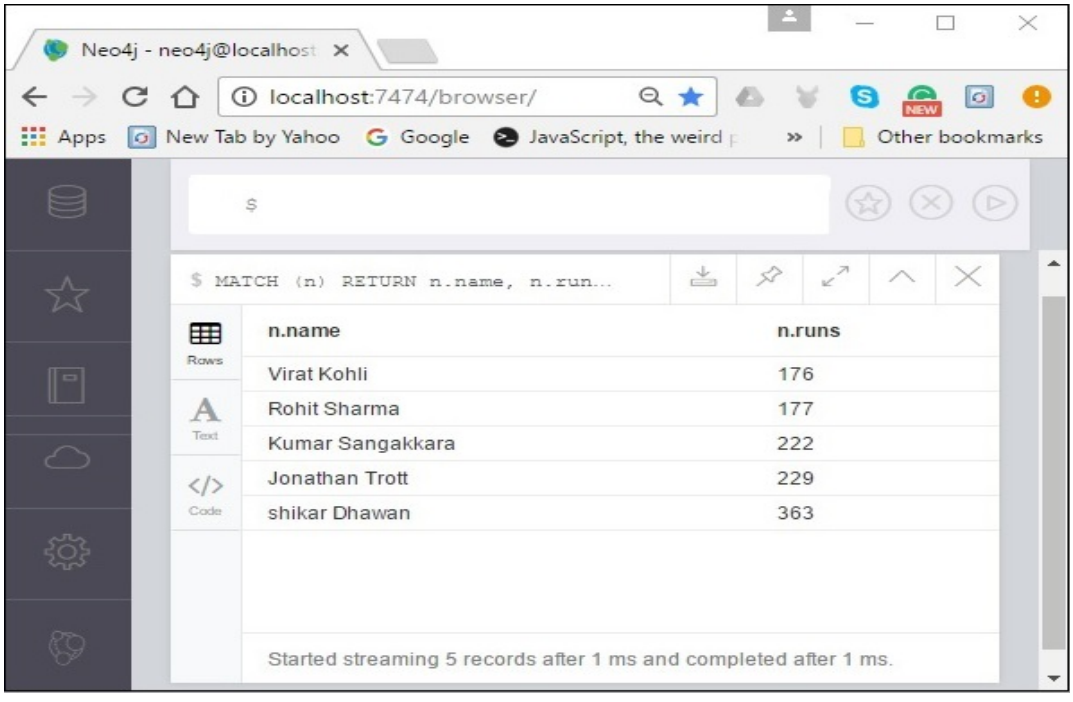
Before proceeding with the example, create 5 nodes in Neo4j database as shown below.

```
CREATE(Dhawan:player{name:"shikar Dhawan", YOB: 1985, runs:363,
country: "India"})
CREATE(Jonathan:player{name:"Jonathan Trott", YOB:1981, runs:229,
country:"South Africa"})
CREATE(Sangakkara:player{name:"Kumar Sangakkara", YOB:1977, runs:222,
country:"Srilanka"})
CREATE(Rohit:player{name:"Rohit Sharma", YOB: 1987, runs:177,
country:"India"})
CREATE(Virat:player{name:"Virat Kohli", YOB: 1988, runs:176, country
```

Following is a sample Cypher Query which returns the above created nodes in the order of the runs scored by the player using the ORDERBY clause.

```
MATCH (n)
RETURN n.name, n.runs
ORDER BY n.runs
```

Result



2. Ordering Nodes by Multiple Properties

You can arrange the nodes based on multiple properties using **ORDEYBY** clause.

Syntax

Following is the syntax to arrange nodes by multiple properties using the ORDERBY clause.

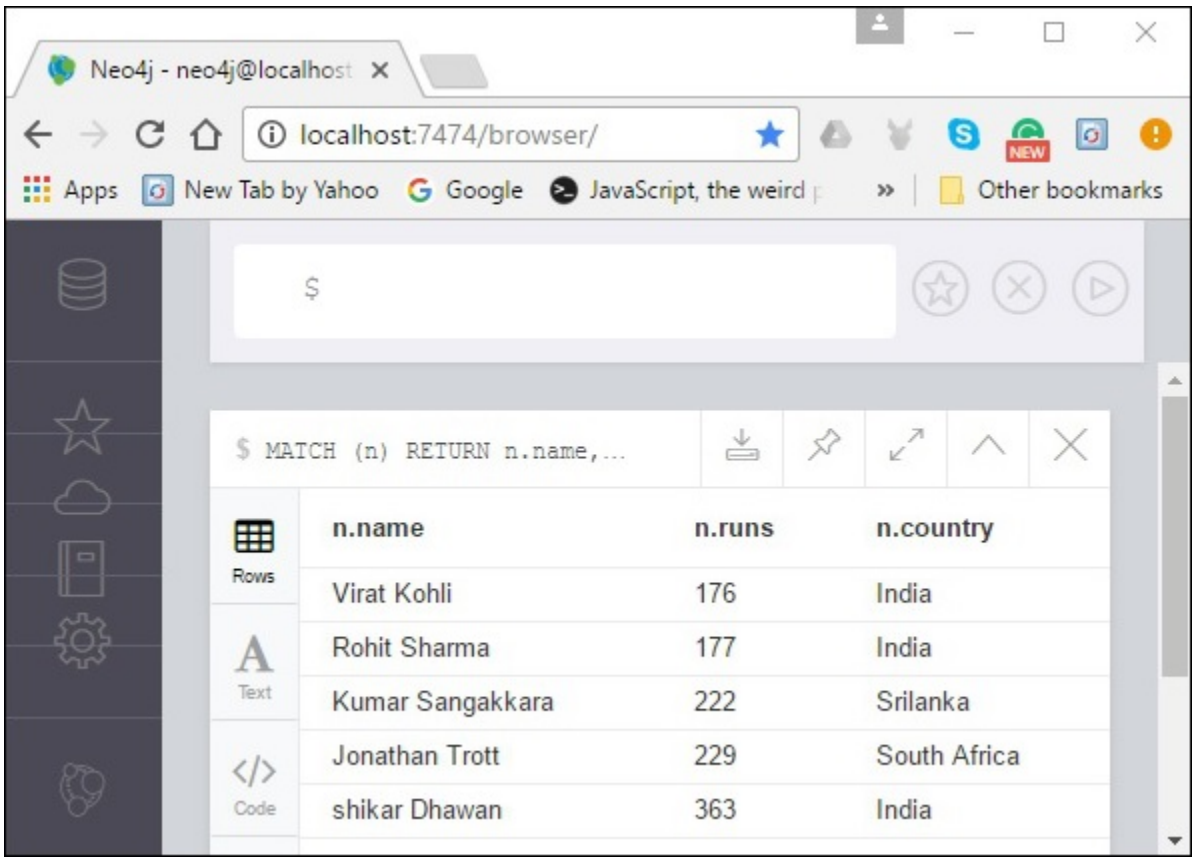
```
MATCH (n)
RETURN n
ORDER BY n.age, n.name
```

Example

Following is a sample Cypher Query which arranges the nodes created earlier in this chapter based on the properties - runs and country.

```
MATCH (n)
RETURN n.name, n.runs, n.country
ORDER BY n.runs, n.country
```

Result



3. Ordering Nodes by Descending Order

You can arrange the nodes in a database in a descending order using the **ORDERBY** clause.

Syntax

Following is the syntax to arrange the nodes in a database.

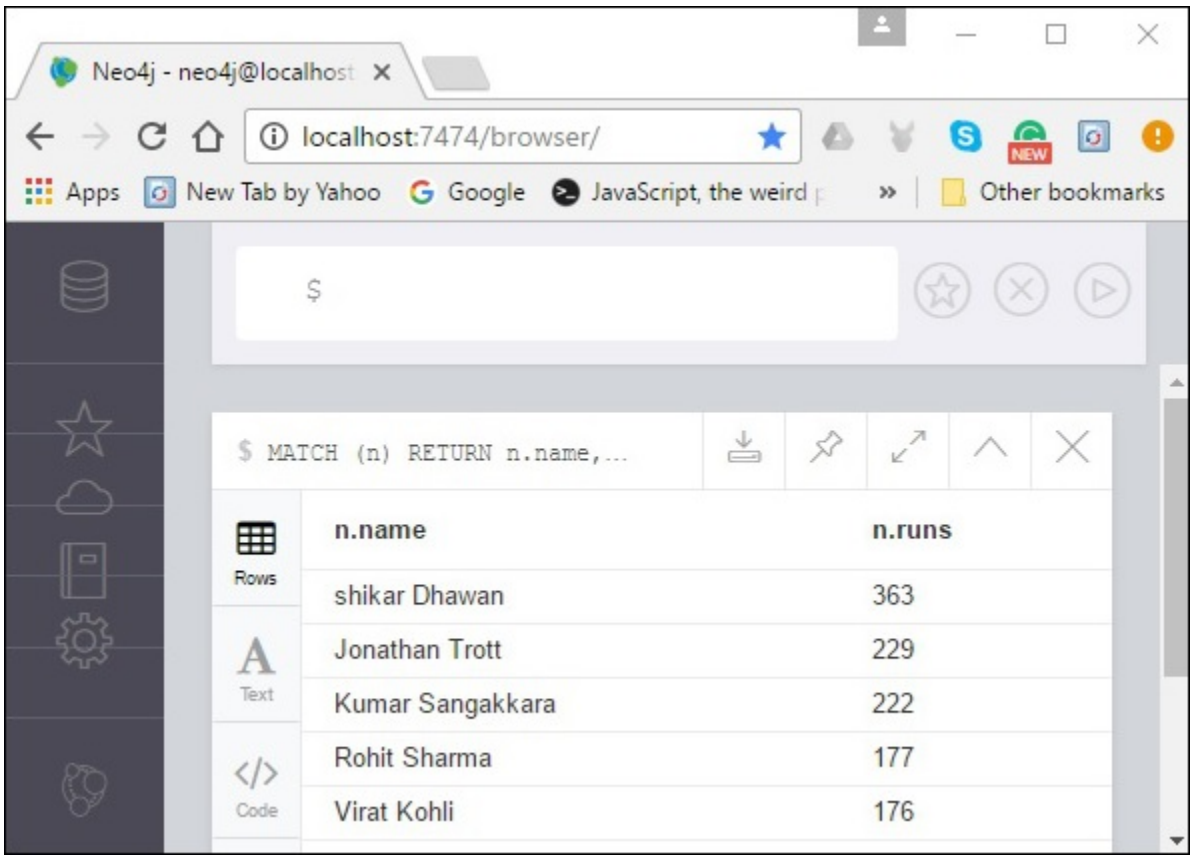
```
MATCH (n)
RETURN n
ORDER BY n.name DESC
```

Example

Following is a sample Cypher Query which arranges the nodes in a database in a descending order using the ORDERBY clause.

```
MATCH (n)
RETURN n.name, n.runs
ORDER BY n.runs DESC
```

Result



Neo4j - Limit Clause

1. The **limit** clause is used to limit the number of rows in the output.

Syntax

Following is the syntax of the LIMIT clause.

```
MATCH (n)
RETURN n
ORDER BY n.name
LIMIT 3
```

Example

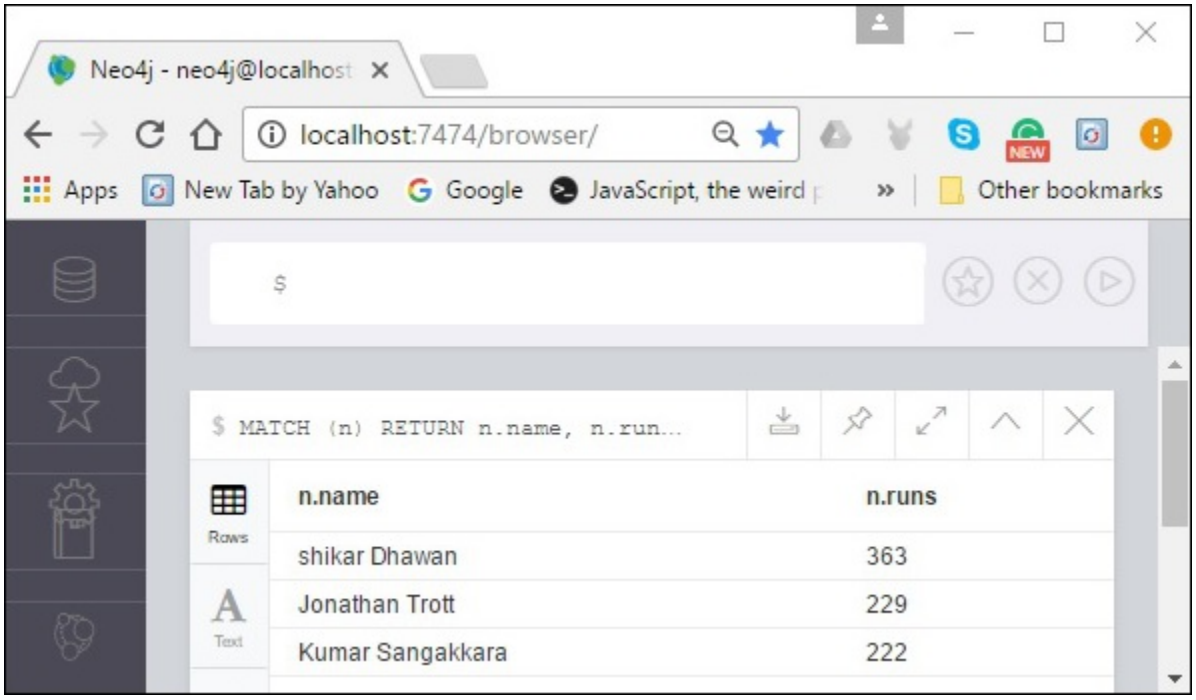
Before proceeding with the example, create 5 nodes in the Neo4j database as shown below.

```
CREATE(Dhawan:player{name:"shikar Dhawan", YOB: 1985, runs:363,
country: "India"})
CREATE(Jonathan:player{name:"Jonathan Trott", YOB:1981, runs:229,
country:"South Africa"})
CREATE(Sangakkara:player{name:"Kumar Sangakkara", YOB:1977, runs:222,
country:"Srilanka"})
CREATE(Rohit:player{name:"Rohit Sharma", YOB: 1987, runs:177,
country:"India"})
CREATE(Virat:player{name:"Virat Kohli", YOB: 1988, runs:176, count
```

Following is a sample Cypher Query which returns the nodes created above in a descending order and limits the records in the result to 3.

```
MATCH (n)
RETURN n.name, n.runs
ORDER BY n.runs DESC
LIMIT 3
```

Result



2. Limit with expression

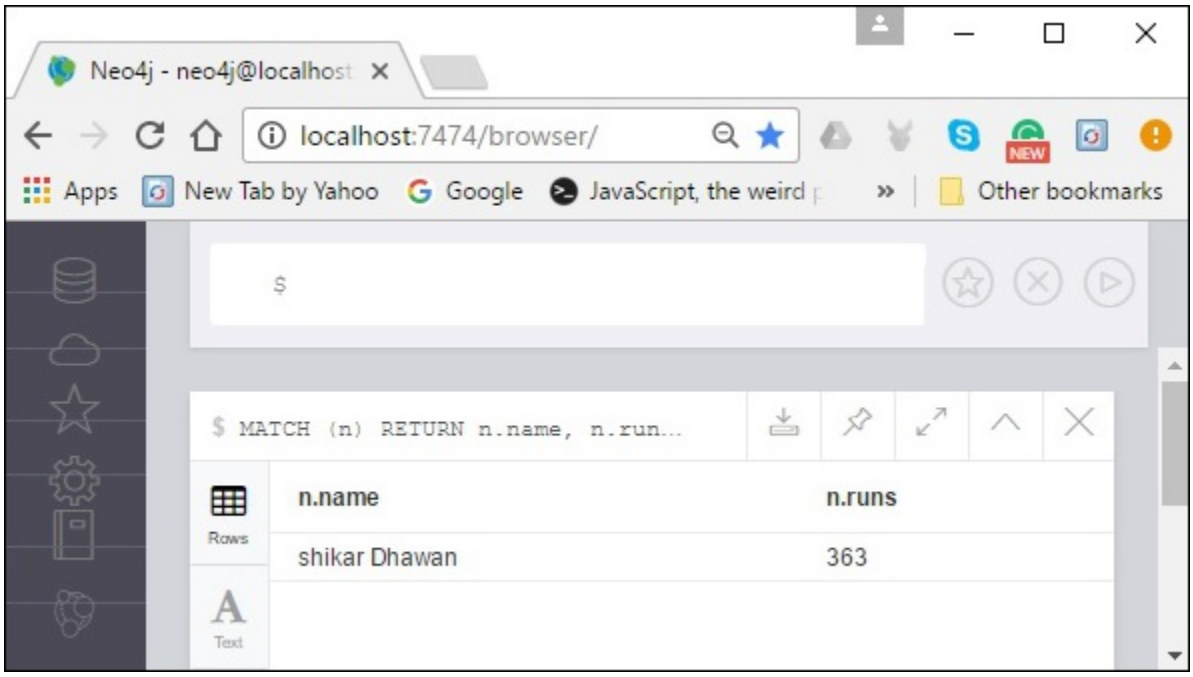
You can also use the LIMIT clause with expression.

Example

Following is a sample Cypher Query which limits the records using an expression.

```
MATCH (n)
RETURN n.name, n.runs
ORDER BY n.runs DESC
LIMIT toInt(3 * rand())+ 1
```

Result



Neo4j - Skip Clause

1. The SKIP clause is used to define from which row to start including the rows in the output.

Example

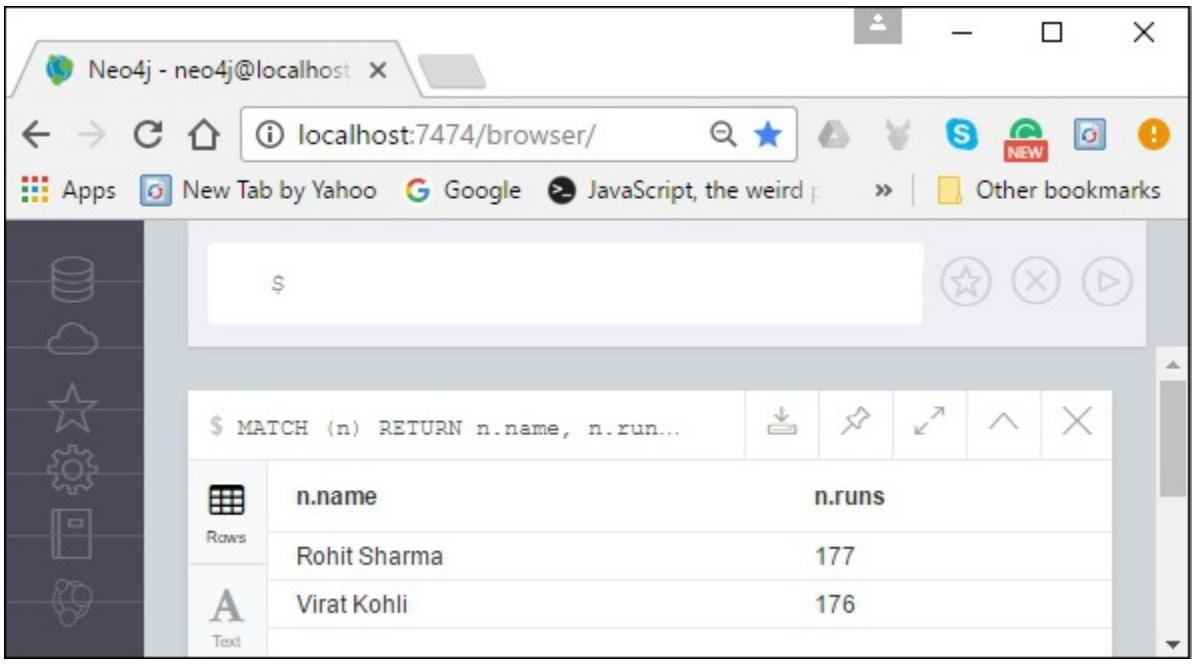
Before proceeding with the example, create 5 nodes as shown below.

```
CREATE(Dhawan:player{name:"shikar Dhawan", YOB: 1985, runs:363,
country: "India"})
CREATE(Jonathan:player{name:"Jonathan Trott", YOB:1981, runs:229,
country:"South Africa"})
CREATE(Sangakkara:player{name:"Kumar Sangakkara", YOB:1977, runs:222,
country:"Srilanka"})
CREATE(Rohit:player{name:"Rohit Sharma", YOB: 1987, runs:177,
country:"India"})
CREATE(Virat:player{name:"Virat Kohli", YOB: 1988, runs:176, countr
```

Following is a sample Cypher Query which returns all the nodes in the database skipping the first 3 nodes.

```
MATCH (n)
RETURN n.name, n.runs
ORDER BY n.runs DESC
SKIP 3
```

Result



2. Skip Using Expression

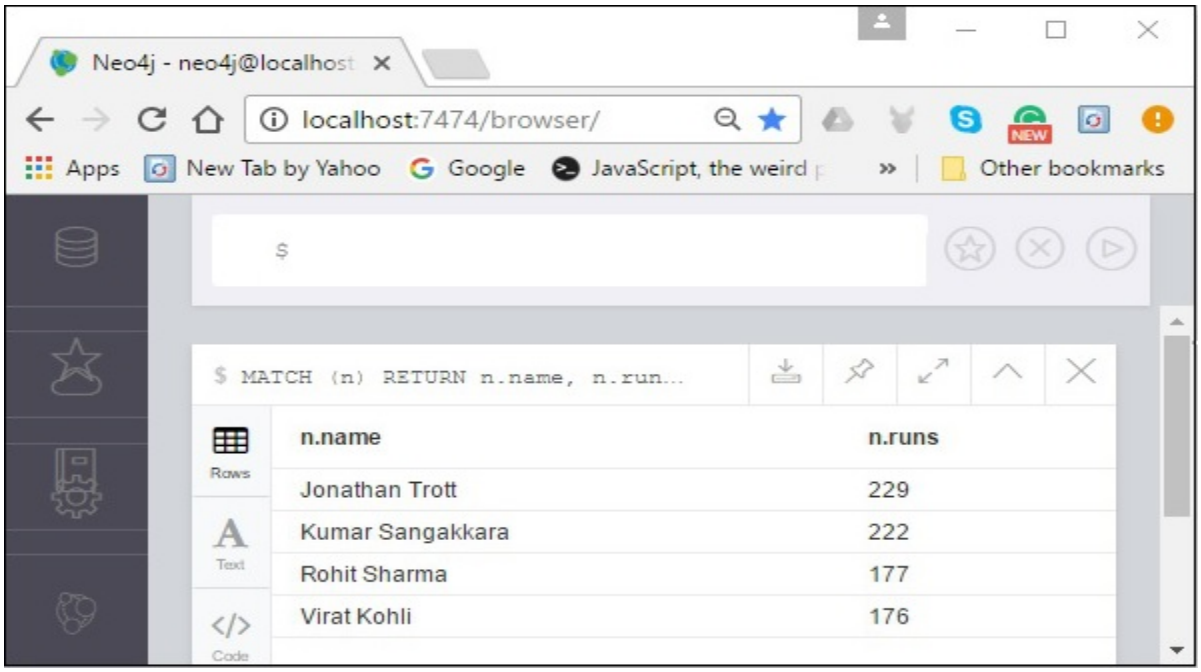
You can skip the records of a result using an expression.

Example

Following is a sample Cypher Query which uses the SKIP clause with an expression.

```
MATCH (n)
RETURN n.name, n.runs
ORDER BY n.runs DESC
SKIP toInt (2*rand())+ 1
```

Result



Neo4j - With Clause

You can chain the query arts together using the WITH clause.

Syntax

Following is the syntax of the WITH clause.

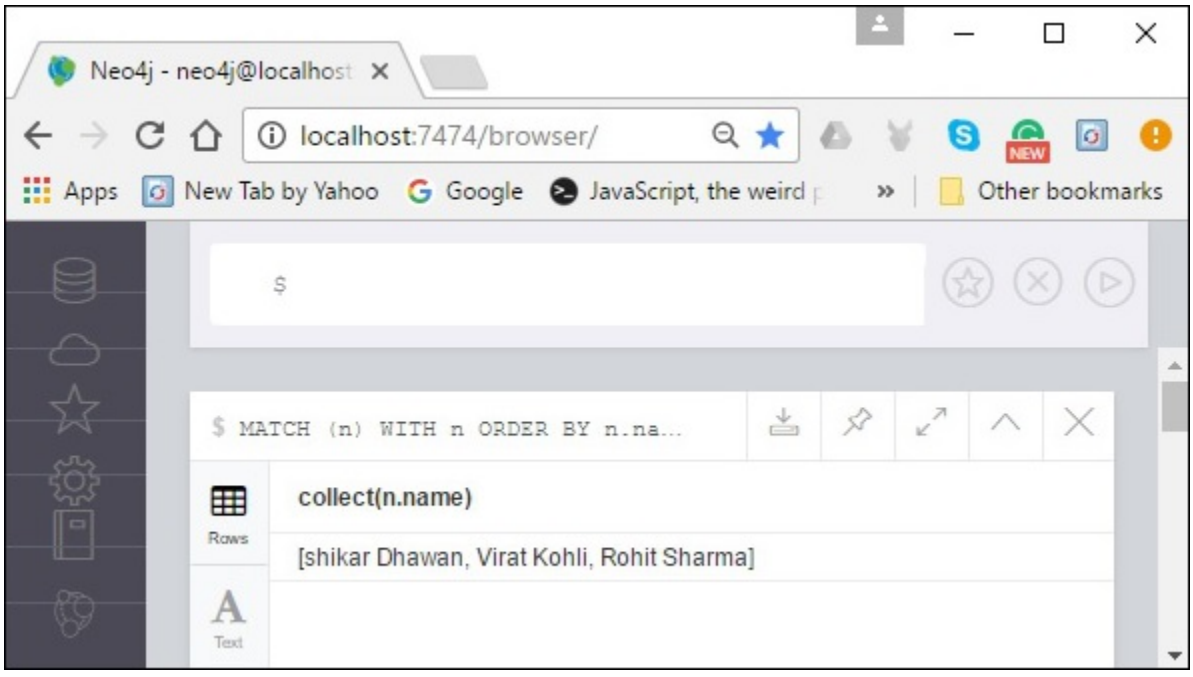
```
MATCH (n)
WITH n
ORDER BY n.property
RETURN collect(n.property)
```

Example

Following is a sample Cypher Query which demonstrates the usage of the WITH clause.

```
MATCH (n)
WITH n
ORDER BY n.name DESC LIMIT 3
RETURN collect(n.name)
```

Result



Neo4j - Unwind Clause

The unwind clause is used to unwind a list into a sequence of rows.

Example

Following is a sample Cypher Query which unwinds a list.

```
UNWIND [a, b, c, d] AS x
RETURN x
```

❖ **Neo4j CQL Functions**

1. String Function
2. Aggregation Function

Neo4j - String Functions

String Functions List

Following is the list of prominent String functions in Neo4j.

Sr.No	Function & Description
1	**UPPER** It is used to change all letters into upper case letters.
2	**LOWER** It is used to change all letters into lower case letters.
3	**SUBSTRING** It is used to get substring of a given String.
4	**Replace** It is used to replace a substring with a given substring of a String.

Neo4j - Aggregation Function

Like SQL, Neo4j CQL has provided some aggregation functions to use in RETURN clause. It is similar to GROUP BY clause in SQL.

We can use this RETURN + Aggregation Functions in MATCH command to work on a group of nodes and return some aggregated value.

AGGREGATION Functions List

Following is the list of aggregation functions in Neo4j.

Sr.No	Function & Description
1	**COUNT** It returns the number of rows returned by MATCH command.
2	**MAX** It returns the maximum value from a set of rows returned by MATCH command.
3	**MIN** It returns the minimum value from a set of rows returned by MATCH command.
4	**SUM** It returns the summation value of all rows returned by MATCH command.
5	**AVG** It returns the average value of all rows returned by MATCH command.