



Neo4j est l'une des bases de données faisant partie de la famille NoSQL Graph dont le langage de requêtage est **Cypher Query Language (CQL)** est l'un des plus populaires. Neo4j est écrit en langage Java. Ce TP explique les bases de Neo4j, la programmation Neo4j en Java, et le Spring DATA avec Neo4j. Le TP est divisé en sections telles que :

- Introduction
- Neo4j CQL
- Neo4j CQL Functions
- Neo4j Admin, etc.
- Chacune de ces sections contient des sujets connexes avec des exemples simples et utiles.

## 1. Introduction au NoSQL avec Neo4J

### 1.1 Qu'est-ce qu'une base de données graphe ?

Un graphe est une représentation abstraite d'un ensemble d'objets où certaines paires d'objets sont reliées par des liens. Il est composé de deux éléments : les nœuds (sommets) et les relations (arêtes). La base de données des graphes est une base de données utilisée pour modéliser les données sous forme de graphe. Ici, les nœuds d'un graphe représentent les entités tandis que les relations représentent l'association de ces nœuds.

Neo4j est une base de données graphe très populaire. Il existe d'autres bases de données graphes tels que Oracle NoSQL, OrientDB, HypherGraphDB, GraphBase, InfiniteGraph, et AllegroGraph.

### 1.2 RDBMS Vs Graph Database

Voici de bases

RDBMS	Graph Database
Tables	Graphs
Rows	Nodes
Column et Data	Les propriétés et leurs valeurs
Constraints	Relationships
Joins	Traversal

le tableau qui compare les bases données relationnelles et les de données graphes.

Voici les avantages de Neo4j.

- **Modèle de données flexible** - Neo4j fournit un modèle de données flexible, simple et pourtant puissant, qui peut être facilement modifié en fonction des applications et des industries.



- **Données en streaming**- Neo4j fournit des résultats basés sur des données en temps réel.
- **Haute disponibilité** - Neo4j est hautement disponible pour les applications temps réel des grandes entreprises avec des garanties transactionnelles.
- **Données connectées et semi-structurées** - En utilisant Neo4j, vous pouvez facilement représenter des données connectées et semi-structurées.
- **Récupération facile** - En utilisant Neo4j, vous pouvez non seulement représenter mais aussi facilement récupérer (parcourir/naviguer) des données connectées plus rapidement par rapport à d'autres bases de données.
- **Cypher query language**- Neo4j fournit un langage de requête déclaratif pour représenter le graphe visuellement, en utilisant une syntaxe ascii-art. Les commandes de ce langage sont au format lisible par l'homme et très faciles à apprendre.
- **Pas de jointures** - En utilisant Neo4j, il ne nécessite pas de jointures complexes pour récupérer des données connectées/liées car il est très facile de récupérer les détails de ses nœuds adjacents ou de ses relations sans jointures ou index.

### 1.3 Modèle de données du graphique de propriété Neo4j

La base de données graphe Neo4j suit le modèle de graphique de propriété pour stocker et gérer ses données. Voici les principales caractéristiques du modèle de propriété de graphe :

- Le modèle représente les données en nœuds, relations et propriétés
- Les propriétés sont des paires de valeurs clés
- Les nœuds sont représentés par un cercle et les relations par des touches fléchées
- Les relations ont des orientations : Unidirectionnelles et bidirectionnelles
- Chaque relation contient un nœud de départ "Start Node" ou "From Node" et "To Node" ou "End Node".
- Les nœuds et les relations contiennent tous deux des propriétés
- Les relations relient les nœuds

Dans le modèle de données des graphes de propriété, les relations doivent être directionnelles. Si nous essayons de créer des relations sans direction, un message d'erreur s'affiche. Dans Neo4j aussi, les relations doivent être directionnelles. Si nous essayons de créer des relations sans direction, alors Neo4j lancera un message d'erreur disant que "Les relations doivent être directionnelles".

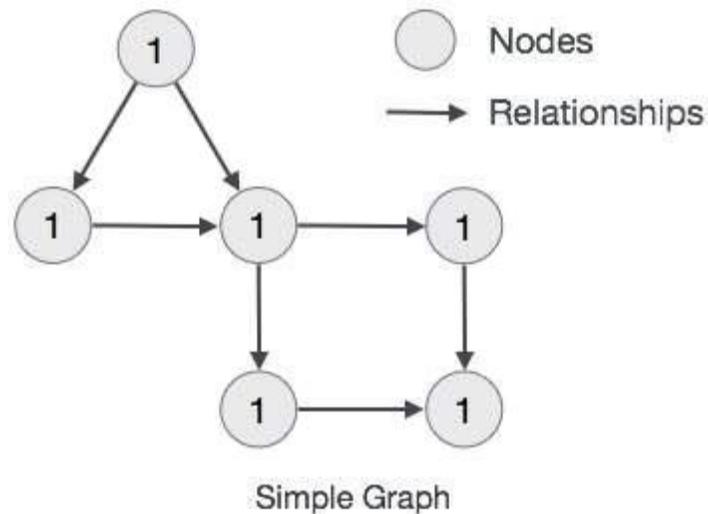
La base de données graphe de Neo4j stocke toutes ses données dans Nodes and Relationships. Nous n'avons besoin d'aucune base de données RRBMS supplémentaire ni d'aucune base de données SQL pour stocker les données de la base Neo4j. Elle stocke ses données en termes de Graphes dans son format natif.

Neo4j utilise Native GPE (Graph Processing Engine) pour travailler avec son format de stockage de graphes natifs.

Les principales composantes du modèle de données de la base de données graphe sont :

- Nœuds
- Relations
- Propriétés

Voici un exemple simple de graphe de propriété.



## 2 Blocs de construction

La base de données graphe Neo4j comporte les éléments suivants :

- Nœuds
- Propriétés
- Relations
- Labels
- Navigateur de données

### 2.1 Node

Le nœud est une unité fondamentale d'un graphe. Il contient des propriétés avec des paires clé-valeur, comme indiqué dans l'image suivante.



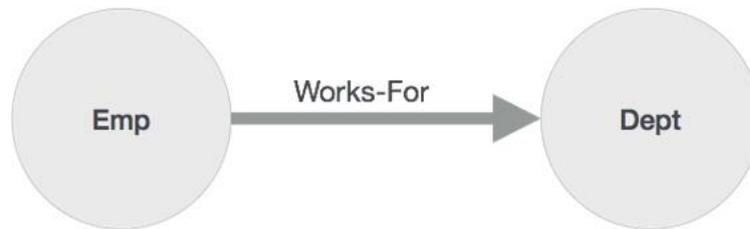
Ici, le nom du nœud = "Employee" et il contient un ensemble de propriétés en tant que paires clé-valeur.

### 2.2 Properties

La propriété est une paire de clés-valeurs (`Key=Value`) pour décrire les valeurs contenues dans nœuds et les relations des graphes. Où la clé est une chaîne de caractère (String) et la valeur peut être représentée en utilisant n'importe quel type de données Neo4j.

### 2.3 Relationships

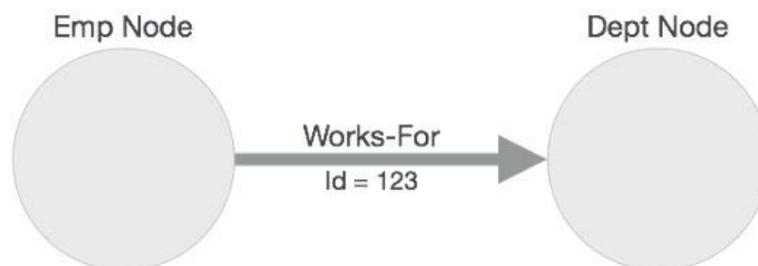
Les relations sont un autre élément majeur d'une base de données graphe. Elle relie deux nœuds comme le montre la figure suivante.



Ici, Emp et Dept sont deux nœuds différents. "WORKS\_FOR" est une relation entre les nœuds Emp et Dept. Comme elle l'indique, la flèche reliant Emp à Dept. Chaque relation contient un nœud de départ et un nœud d'arrivée. Ici, "Emp" est un nœud de départ et "Dept" un nœud d'arrivée.

Comme cette flèche représente une relation entre le nœud "Emp" et le nœud "Dept", cette relation est connue sous le nom de "relation entrante" vers le nœud "Dept" et de "relation sortante" vers le nœud "Emp".

Comme les nœuds, les relations peuvent également contenir des propriétés en tant que paires clé-valeur.



Ici, la relation "WORKS\_FOR" a une propriété comme paire clé-valeur.

## 2.4 Labels

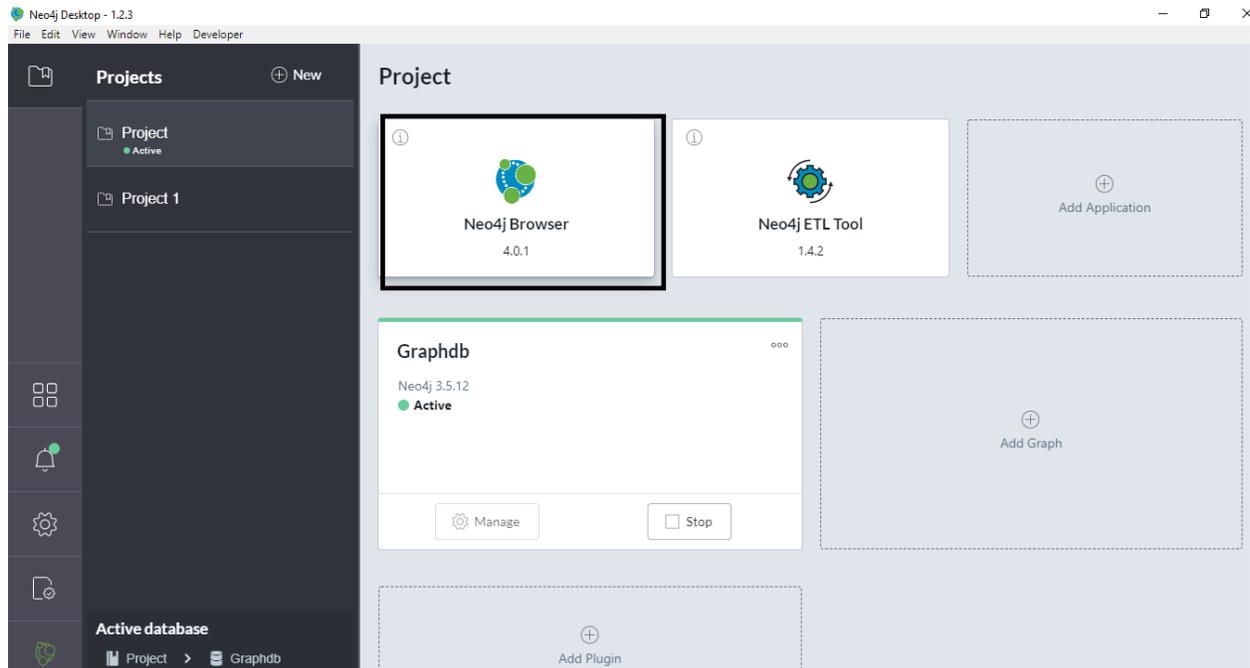
Le label associe un nom commun à un ensemble de nœuds ou de relations. Un nœud ou une relation peut contenir une ou plusieurs étiquettes. Nous pouvons créer de nouvelles étiquettes pour des nœuds ou des relations existants. Nous pouvons supprimer les étiquettes existantes des nœuds ou relations existants. Dans le diagramme précédent, nous pouvons observer qu'il y a deux nœuds. Le nœud de gauche a un Label : "Emp" et le nœud du côté droit a un Label : "Dept". La relation entre ces deux nœuds a également un Label : "WORKS\_FOR".

**Note** - Neo4j stocke les données dans les propriétés des nœuds ou des relations.



## 2.5 Neo4j Data Browser

Une fois que nous avons installé Neo4j, nous pouvons accéder au navigateur de données Neo4j en utilisant via :



Ou via le navigateur web <http://localhost:7474/browser/>

## 3 Cypher Query Language

CQL signifie Cypher Query Language. Tout comme la base de données Oracle possède le langage de requête SQL, Neo4j possède CQL comme langage de requête.

### Requête de lecture

Clauses de lecture	Utilisation
MATCH	Cette clause est utilisée pour rechercher les données selon un schéma précis.
OPTIONAL MATCH	C'est la même chose que la clause MATCH, la seule différence étant qu'elle peut utiliser des valeurs NULL en cas de parties manquantes du modèle.
WHERE	Cette clause est utilisée pour ajouter du contenu aux requêtes CQL.
START	Cette clause est utilisée pour trouver les points de départ à travers les anciens index.



LOAD CSV	Cette clause est utilisée pour importer des données à partir de fichiers CSV.
----------	---

## Requête d'écriture

Clause d'écriture	Utilisation
CREATE	Cette clause est utilisée pour créer des nœuds, des relations et des propriétés.
MERGE	Cette clause permet de vérifier si le pattern spécifié existe dans le graphe. Si ce n'est pas le cas, elle crée le modèle.
SET	Cette clause est utilisée pour mettre à jour les étiquettes sur les nœuds, les propriétés sur les nœuds et les relations.
DELETE	Cette clause est utilisée pour supprimer les nœuds et les relations ou les chemins, etc. du graphe.
REMOVE	Cette clause est utilisée pour supprimer les propriétés et les éléments des nœuds et des relations.
FOREACH	Cette clause est utilisée pour mettre à jour les données au sein d'une liste.
CREATE UNIQUE	En utilisant les clauses CREATE et MATCH, vous pouvez obtenir un modèle unique en faisant correspondre le modèle existant et en créant le modèle manquant.
Load CSV	En utilisant Load CSV, vous pouvez importer des données à partir de fichiers .csv.

## Voici les clauses générales de Neo4j Cypher Query Language -

Clause d'écriture	Utilisation
RETURN	Cette clause est utilisée pour définir ce qui doit être inclus dans l'ensemble des résultats de la requête.



ORDER BY	Cette clause est utilisée pour organiser la sortie d'une requête dans l'ordre. Elle est utilisée avec les clauses <b>RETURN</b> ou <b>WITH</b> .
LIMIT	Cette clause est utilisée pour limiter les lignes du résultat à une valeur spécifique.
SKIP	Cette clause est utilisée pour définir à partir de quelle ligne il faut commencer à inclure les lignes dans la sortie.
WITH	Cette clause est utilisée pour enchaîner les parties de la requête.
UNWIND	Cette clause est utilisée pour développer une liste en une séquence de lignes.
UNION	Cette clause est utilisée pour combiner le résultat de plusieurs requêtes.
CALL	Cette clause est utilisée pour invoquer une procédure déployée dans la base de données.

## Fonctions CQL

Clause d'écriture	Utilisation
String	Ils sont habitués à travailler avec les String littéraux.
Aggregation	Ils sont utilisés pour effectuer certaines opérations d'agrégation sur les résultats de la requête CQL.
Relationship	Ils sont utilisés pour obtenir des détails sur les relations telles que le nœud de départ, le nœud d'arrivée, etc.

## Types de données CQL de Neo4j



Type de données	Utilisation
Boolean	Il est utilisé pour représenter les littéraux booléens : vrai, faux.
byte	Il est utilisé pour représenter un entier sur 8 bits
short	Il est utilisé pour représenter un entier sur 16 bits
int	Il est utilisé pour représenter un entier sur 32 bits
long	Il est utilisé pour représenter un entier sur 64 bits
float	Il est utilisé pour représenter des nombres à virgule flottante de 32 bits.
double	Il est utilisé pour représenter des nombres à virgule flottante de 64 bits.
char	Il est utilisé pour représenter les caractères de 16 bits.
String	Il est utilisé pour représenter un String

## Operateurs CQL



Type	Utilisation
Opérateur arithmétique	+, -, *, /, %, ^
Opérateur de comparaison	+, <>, <, >, <=, >=
Opérateur boolean	AND, OR, XOR, NOT
String	+
List	+, IN, [X], [X.....Y]
Opérateur d'expression	==
Opération de matching	STARTS WITH, ENDS WITH, CONSTRAINTS

## 4 Création de nœuds

Dans cette section, nous discuterons sur comment :

- Créer un nœud unique
- Créer des nœuds multiples
- Créer un nœud avec une étiquette
- Créer un nœud avec plusieurs étiquettes
- Créer un nœud avec des propriétés
- Retourner un nœud créé



## 4.1 Création d'un simple Nœud

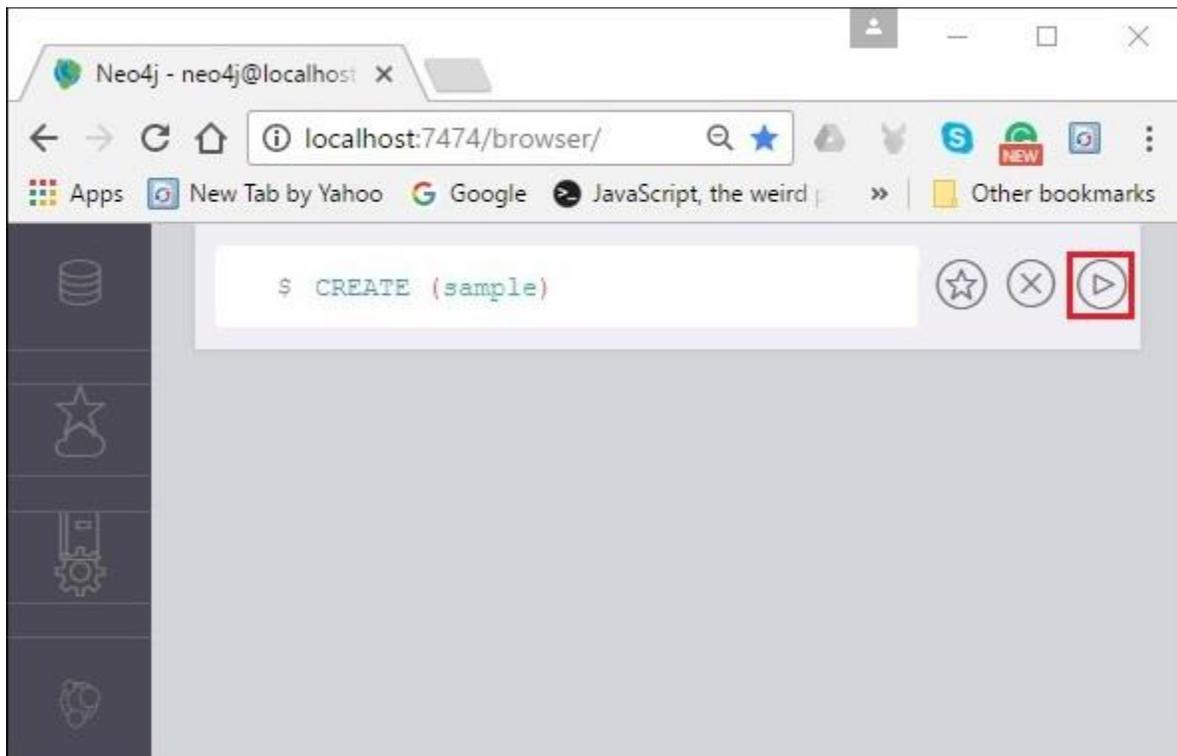
Vous pouvez créer un nœud dans Neo4j en spécifiant simplement le nom du nœud qui doit être créé avec la clause CREATE.

Syntaxe

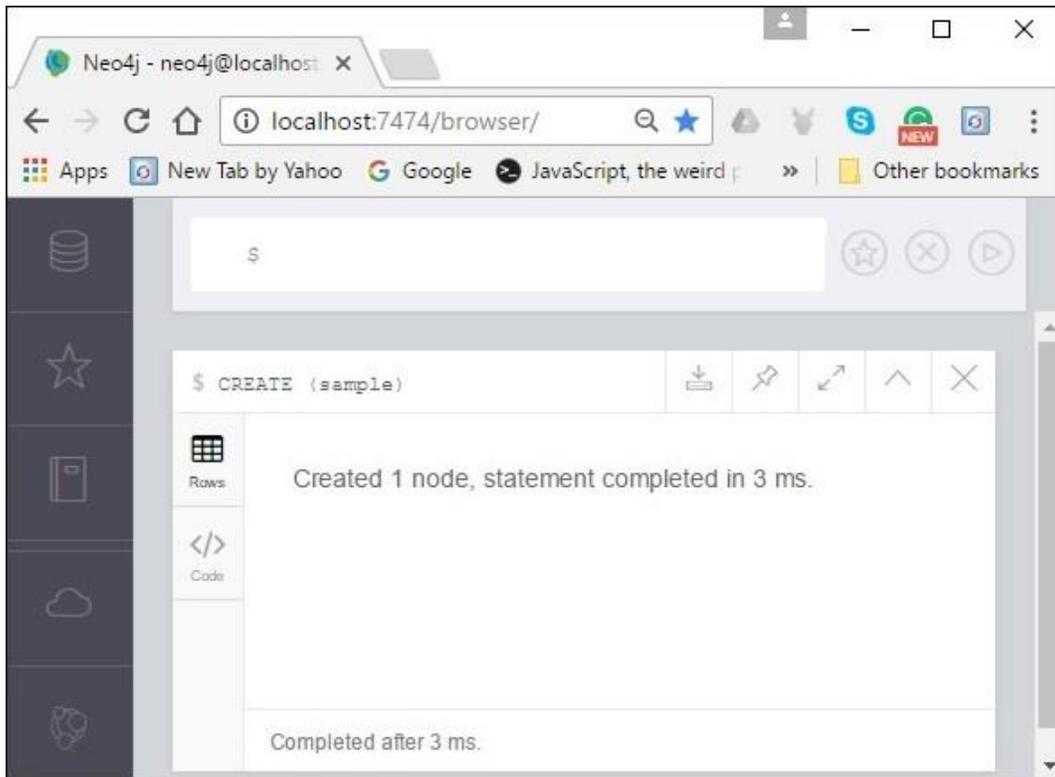
Voici la syntaxe pour créer un nœud en utilisant le langage d'interrogation par CQL est :

```
CREATE (node_name);
```

Copiez et collez la requête souhaitée dans l'invite dollar et appuyez sur le bouton play (pour exécuter la requête) mis en évidence dans la capture d'écran suivante.



Lors de l'exécution, vous obtiendrez le résultat suivant.

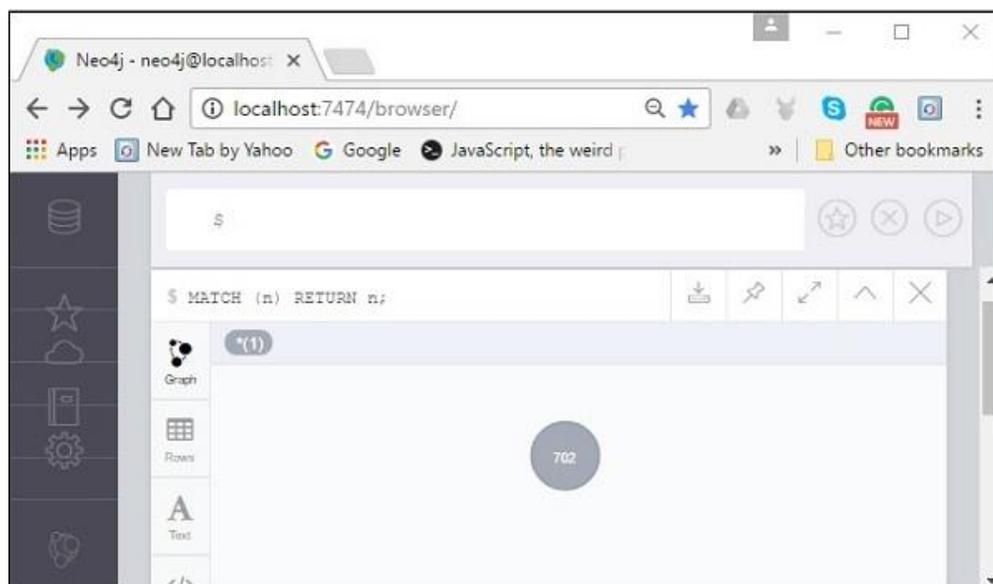


Pour vérifier la création du type de nœud, exécutez la requête suivante dans l'invite de commande.

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

Cette requête renvoie tous les nœuds de la base de données (nous discuterons de cette requête en détail dans les prochains chapitres).

À l'exécution, cette requête affiche le nœud créé comme le montre la capture d'écran suivante.





## 4.2 Création de nœuds multiples

La clause de création de Neo4j CQL est également utilisée pour créer plusieurs nœuds en même temps. Pour ce faire, vous devez passer les noms des nœuds à créer, séparés par une virgule.

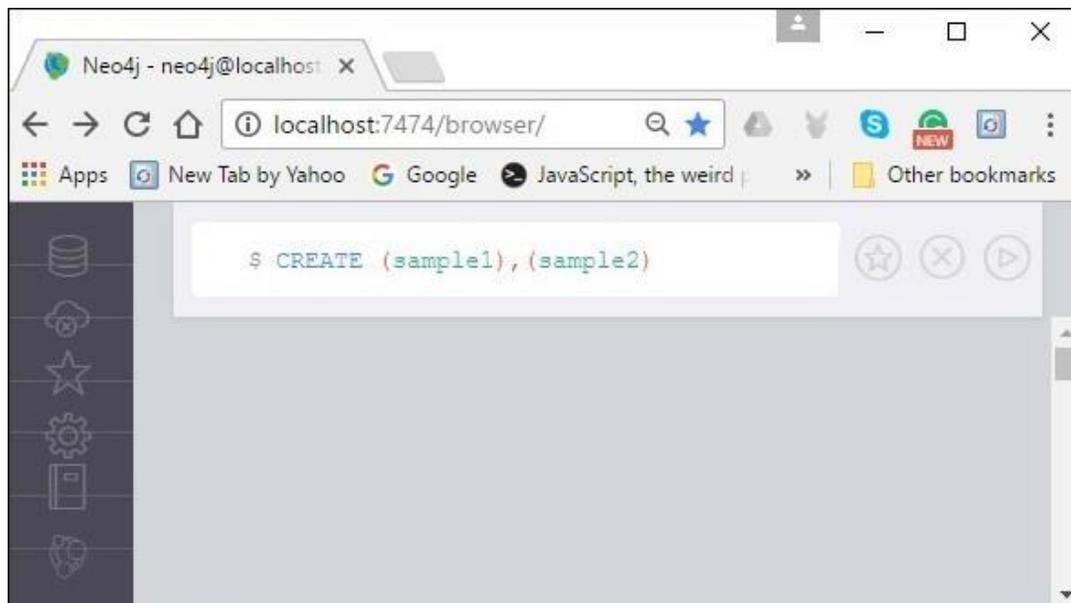
### Syntaxe

Voici la syntaxe pour créer des nœuds multiples en utilisant la clause **CREATE**.

```
CREATE (node1), (node2)
```

Voici un exemple de requête de chiffrement qui crée plusieurs nœuds dans Neo4j.

```
CREATE (sample1), (sample2)
```



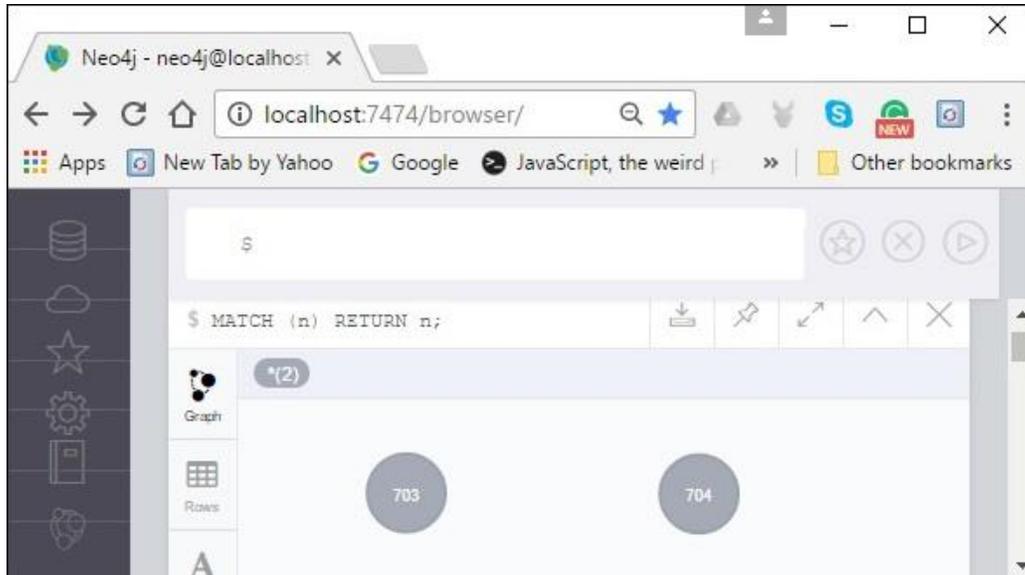
Lors de l'exécution, vous obtiendrez le résultat suivant.





Pour vérifier la création du nœud, tapez et exécutez la requête suivante dans l'invite commande.

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



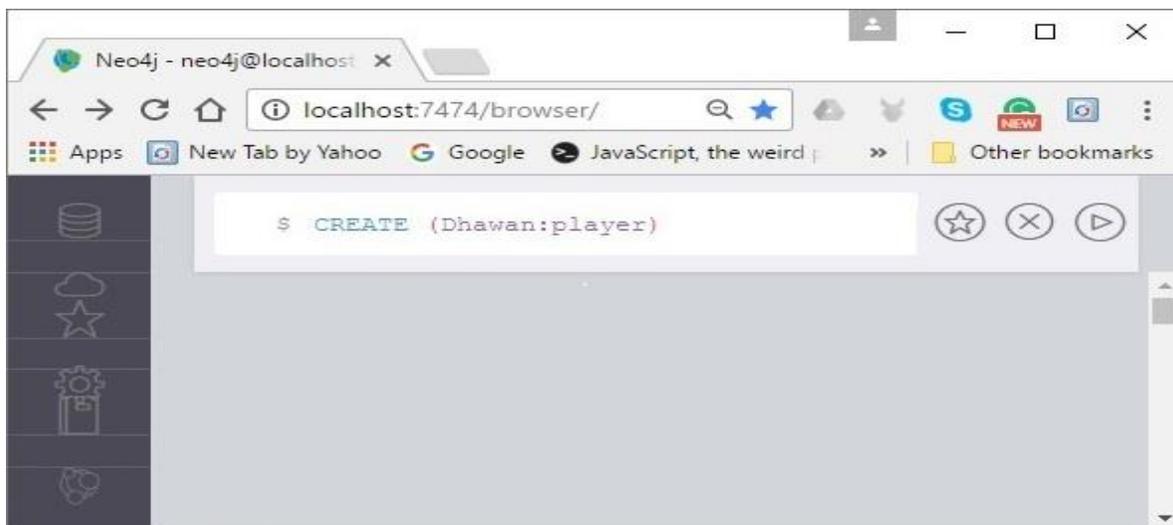
### 4.3 Création de Nœud avec Label

Une étiquette dans Neo4j est utilisée pour regrouper (classer) les nœuds à l'aide d'étiquettes. Vous pouvez créer un label pour un nœud dans Neo4j en utilisant la clause CREATE.

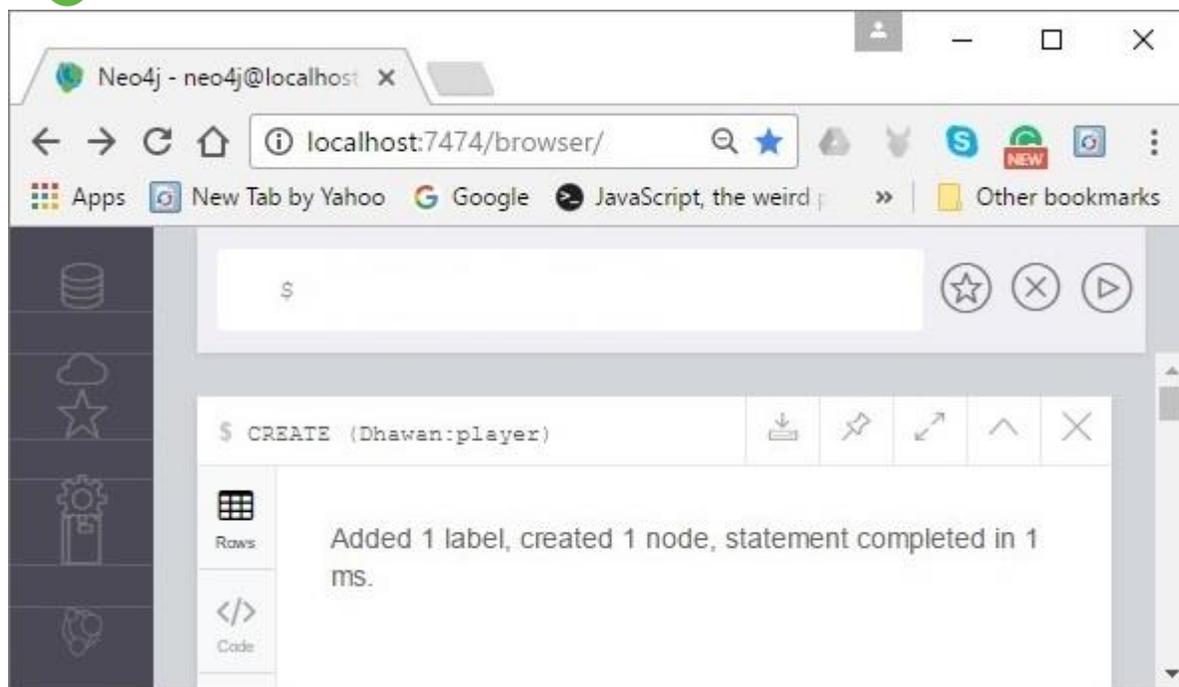
```
CREATE (nœud :label)
```

Voici un exemple de requête de CQL qui crée un nœud avec une étiquette.

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



Lors de l'exécution, vous obtiendrez le résultat suivant.

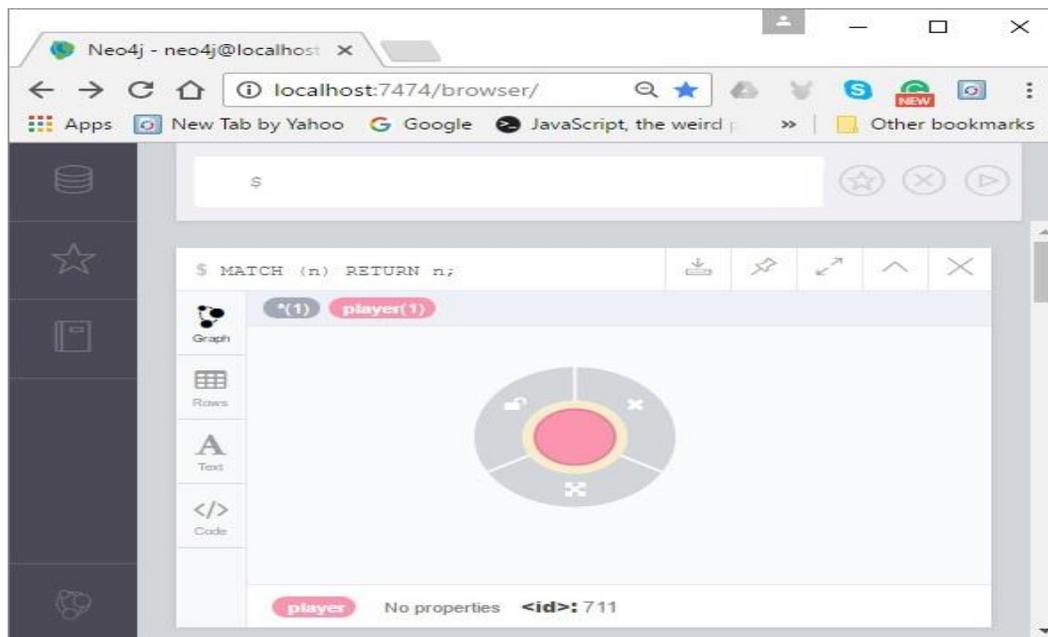


Pour vérifier la création du nœud, tapez et exécutez la requête suivante dans l'invite commande.

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

Cette requête renvoie tous les nœuds de la base de données.

À l'exécution, cette requête affiche le nœud créé comme le montre la capture d'écran suivante.





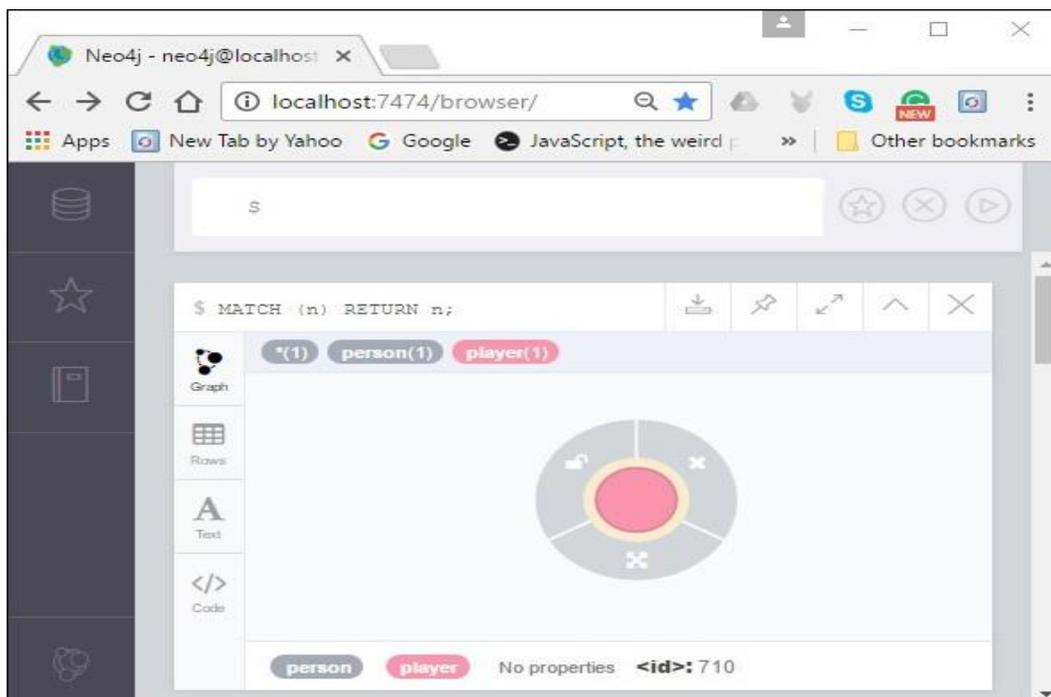
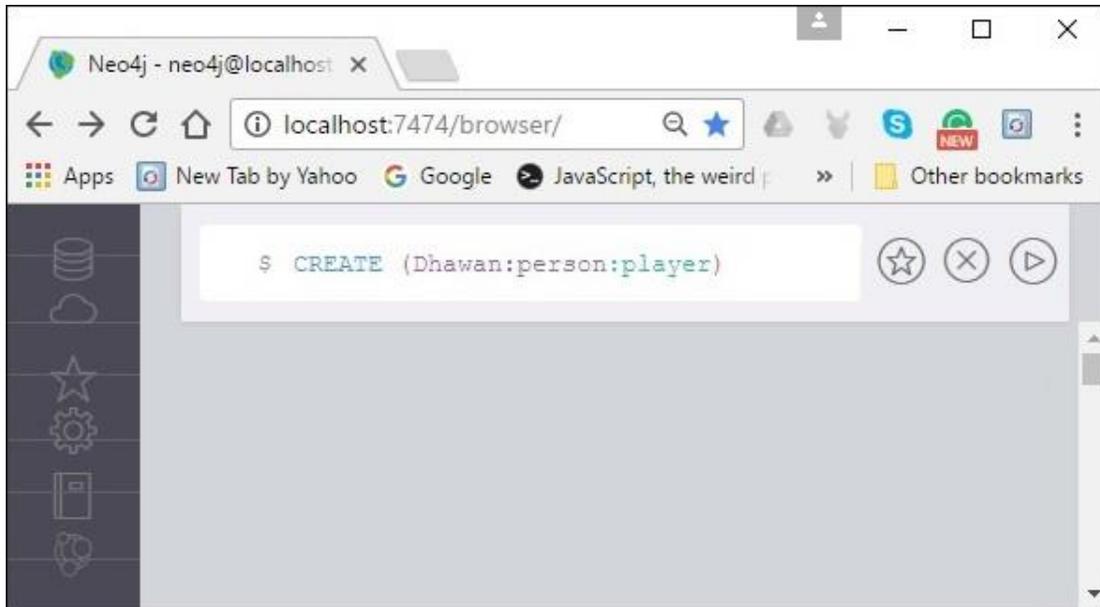
#### 4.4 Création de Nœud avec plusieurs labels

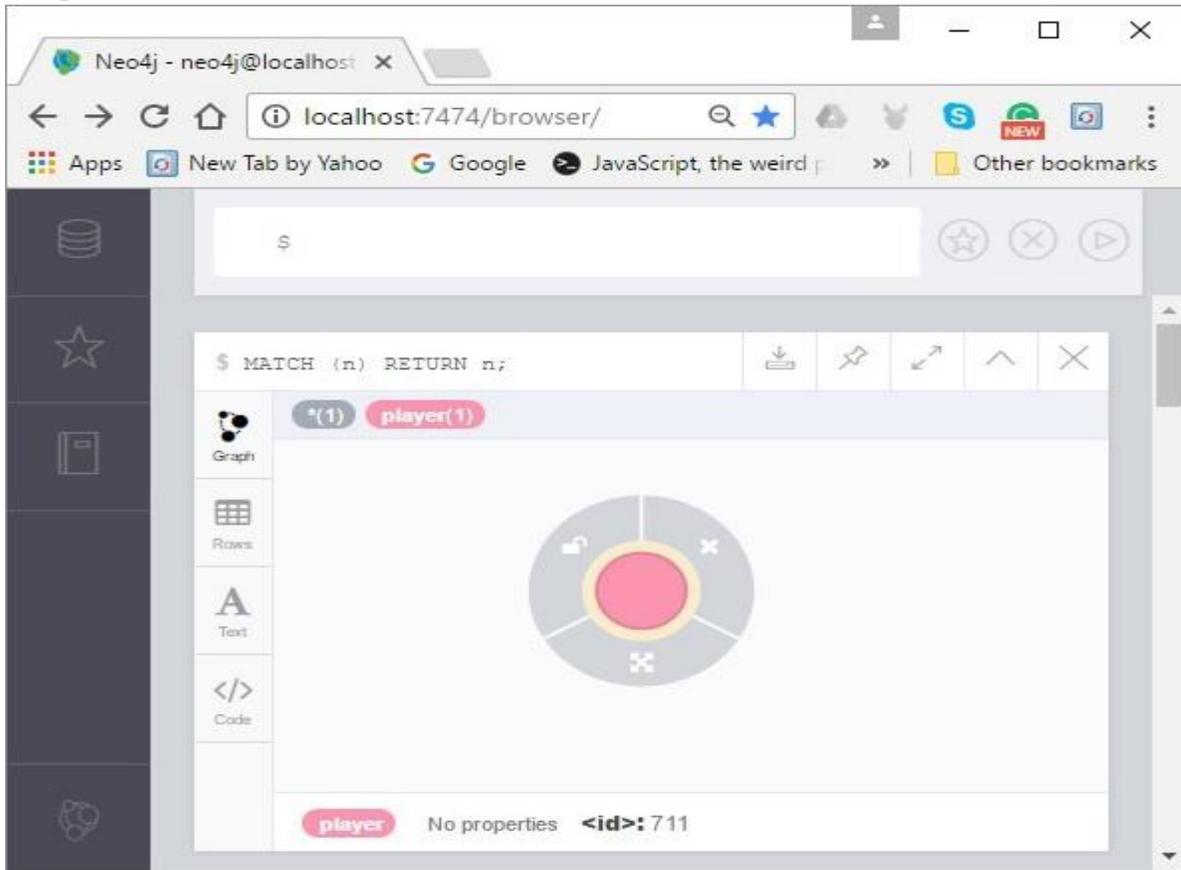
Vous pouvez également créer plusieurs étiquettes pour un même nœud. Vous devez spécifier les étiquettes pour le nœud en les séparant par deux points " : ".

```
CREATE (node1:label1 :label2 : ... :labeln)
```

La commande suivante crée un nœud avec plusieurs étiquettes dans Neo4j.

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





#### 4.5 Création de Nœud avec Label

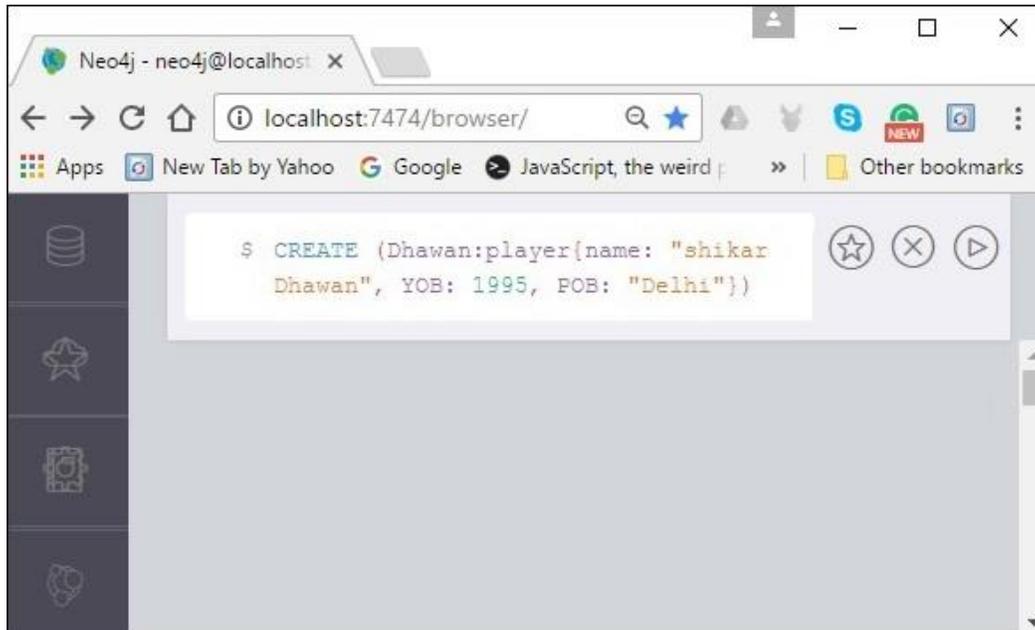
Les propriétés sont les paires clé-valeur à l'aide desquelles un nœud stocke les données. Vous pouvez créer un nœud avec des propriétés en utilisant la clause CREATE. Vous devez spécifier ces propriétés en les séparant par des virgules dans les accolades "{}".

Voici la syntaxe pour créer un nœud avec des propriétés.

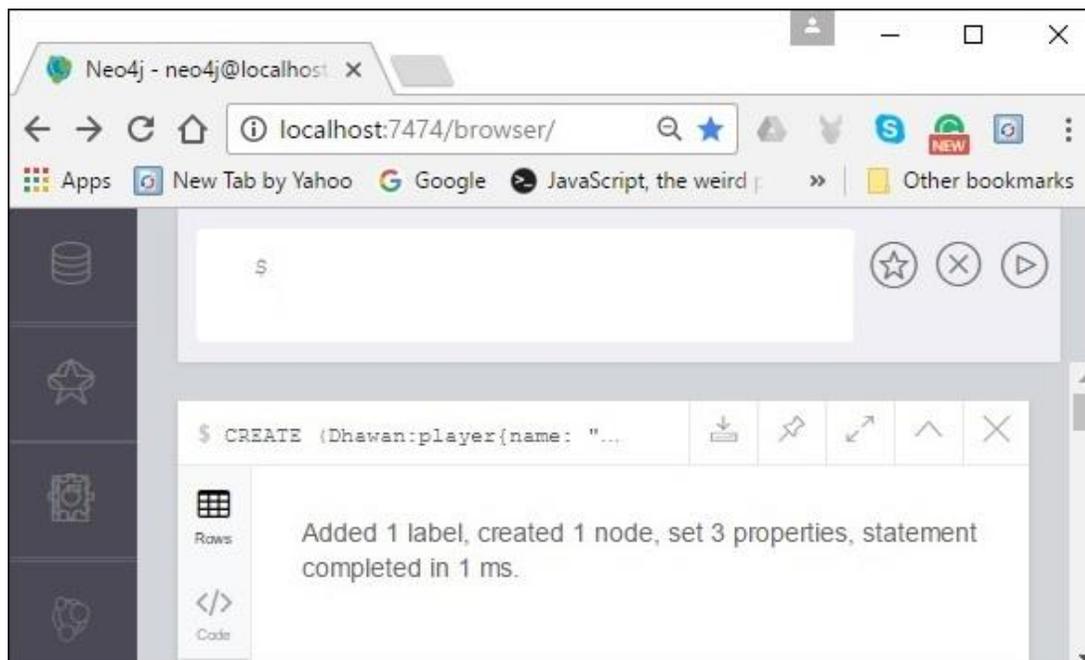
```
CREATE (node:label { key1: value, key2: value, . . . . . })
```

Voici un exemple de requête de chiffrement qui crée un nœud avec des propriétés.

```
CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "De1hi"})
```



Lors de l'exécution, vous obtiendrez le résultat suivant.

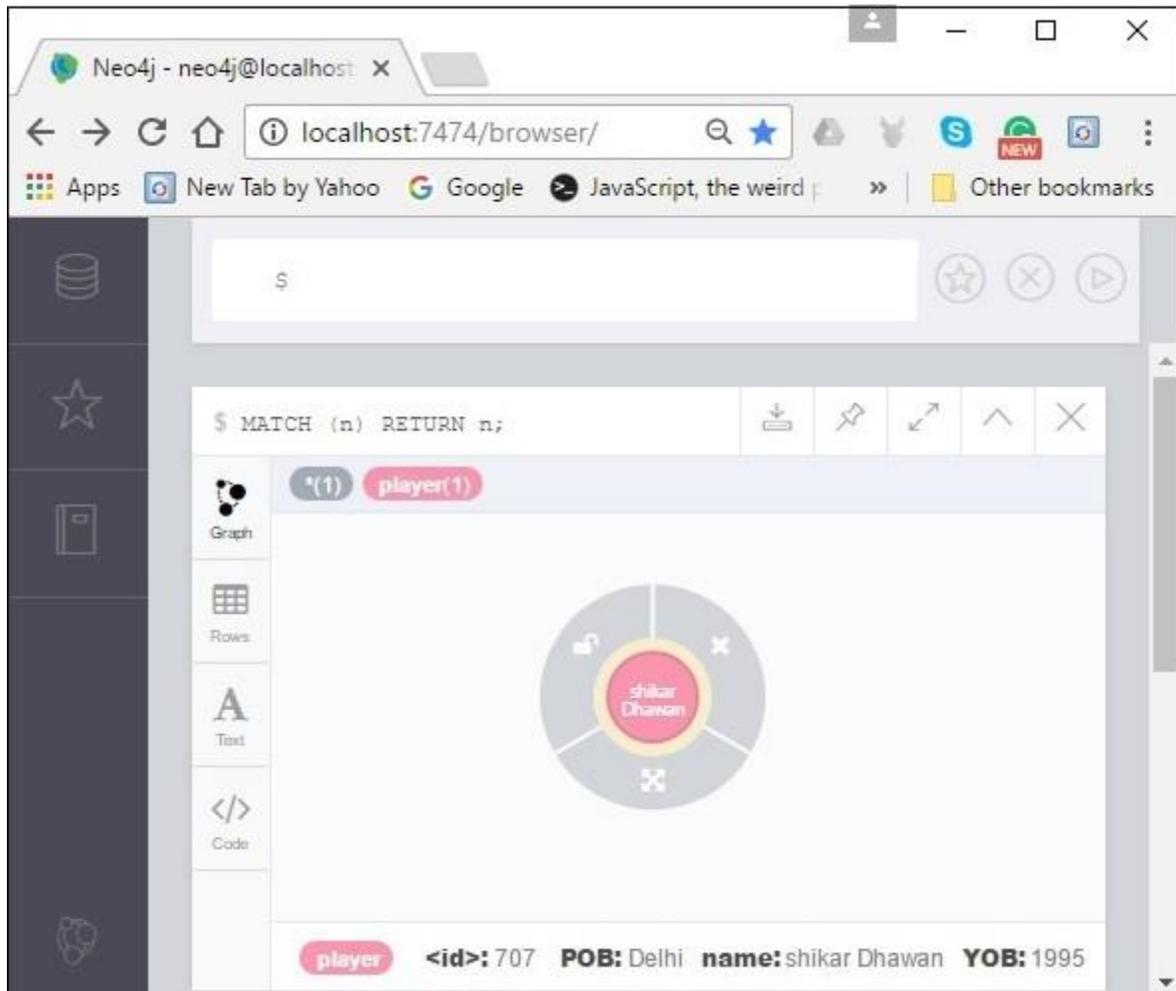


Pour vérifier la création du nœud, tapez et exécutez la requête suivante dans l'invite commande.

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

Cette requête renvoie tous les nœuds de la base de données.

À l'exécution, cette requête affiche le nœud créé comme le montre la capture d'écran suivante.



## 4.6 Création de Nœud avec Label

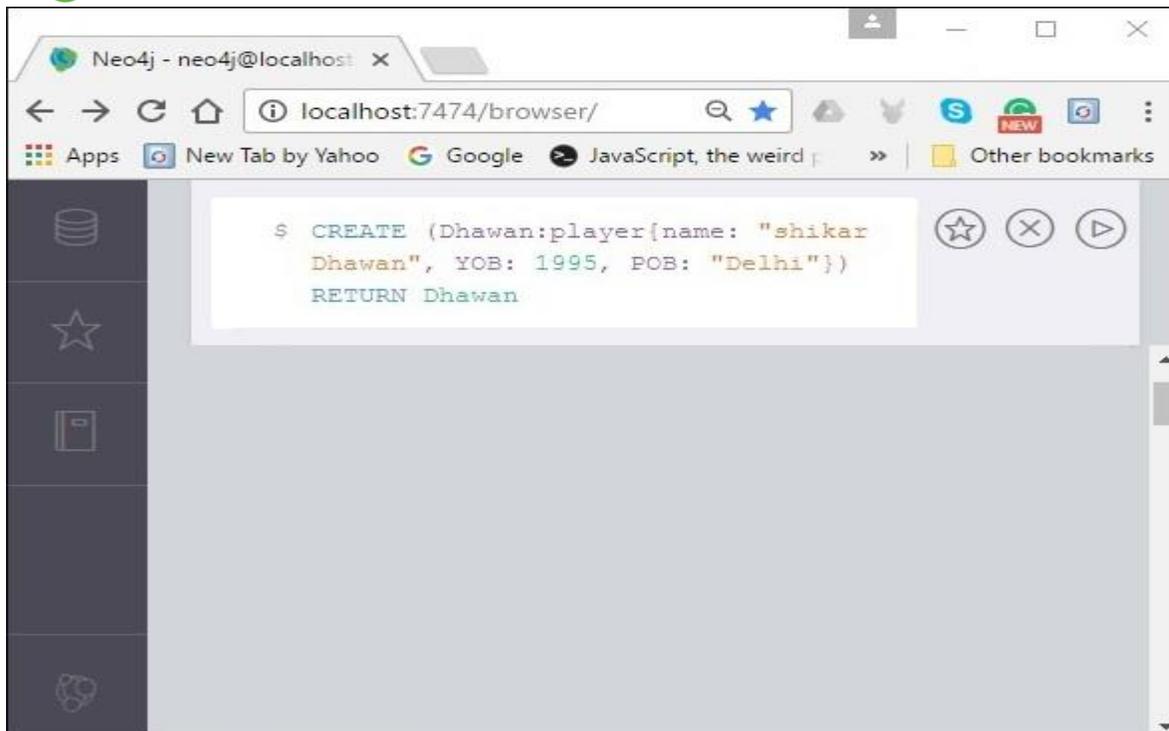
Tout au long du TP, nous avons utilisé la requête **MATCH (n) RETURN n** pour visualiser les nœuds créés. Cette requête renvoie tous les nœuds existants dans la base de données. En plus de cela, nous pouvons utiliser la clause **RETURN** avec **CREATE** pour visualiser le nœud nouvellement créé.

Voici la syntaxe pour retourner un nœud dans Neo4j.

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

Voici un exemple de requête de CQL qui crée un nœud avec des propriétés et le renvoie.

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



Lors de l'exécution, vous obtiendrez le résultat suivant.

