TP – Neo4J – NBA_Graph_DB[1]

Schéma BDD: NBA.Graph.DB

Nous avons une base de données est une **base de données NBA** centrée sur :

- des **joueurs** (PLAYER)
- des **entraîneurs** (COACH)
- des **équipes** (TEAM)
- des **matchs joués entre joueurs et équipes** (PLAYED_AGAINST)
- des **relations hiérarchiques et d'appartenance** (PLAYS_FOR, COACHES_FOR, etc.)
- des **relations sociales** comme les coéquipiers (TEAMMATES)

Nous allons :

1. **Créer les nœuds**

Nous avons donné 3 labels distincts :

- PLAYER
- COACH
- TEAM

Ils contiennent des propriétés pertinentes : nom, âge, taille, poids, numéro, etc.

2. **Créer la relation TEAMMATES : relations orientées en double sens**

Tu écris par exemple :

```
(lebron)-[:TEAMMATES]-> (russell),
(lebron)<-[:TEAMMATES]- (russell),
```

P.S 1 : Neo4J ne gère pas automatiquement les relations bidirectionnelles.
P.S 2 : *Alternative plus propre* : un seul lien (:PLAYER)-[:TEAMMATES]-(:PLAYER) non orienté.

3. **Créer les relations d'encadrement COACHES et COACHES_FOR**

COACHES : entraîneur → joueur

[1] https://github.com/harblaith7/Neo4j-Crash-Course/blob/main/01-initial-data.cypher

COACHES_FOR : entraîneur → équipe

4. **Créer les relations d'appartenance PLAYS_FOR avec propriété salaire**

Nous allons ajouter des propriétés (salaire) sur une relation (Plays_for).

5. **Créer les relations de match PLAYED_AGAINST**

Chaque relation encode des statistiques individuelles *d'un joueur contre une équipe*. Les propriétés forment un mini-enregistrement statistique.

```
(lebron)-[:PLAYED_AGAINST {minutes: 38, points: 32, ...}]-> (memphis)
```

Nous avons **plusieurs relations PLAYED_AGAINST pour les mêmes duels**, car chaque match génère plusieurs entrées.

Cela peut dupliquer l'information *si l'on ne distingue pas les dates ou les IDs de match*.

Pour éviter les conflits futurs : ajouter une propriété game_id ou date.

Exécution

1. Ouvrir l'environnement Neo4J et démarrer la BDD.
2. Pour éviter les doublons, **toujours** commencer par supprimer les données existantes.

Neo4J ne bloque pas la suppression si les nœuds sont liés grâce à « Detach Delete » :

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

3. Activer les contraintes pour éviter des doublons de noms.

```
CREATE CONSTRAINT player_name IF NOT EXISTS
FOR (p:PLAYER)
REQUIRE p.name IS UNIQUE;
```

4. Coller le script CREATE et exécuter.
5. Afficher mon schéma.

```
CALL db.schema.visualization()
```

6. Vérifier que les nœuds ont été créés

```
MATCH (n) RETURN n LIMIT 25;
```

7. Vérifier les types de relations

```
//Voir tous les types de relations existants

CALL db.relationshipTypes();

//Lister toutes les relations PLAYED_AGAINST

MATCH ()-[r:PLAYED_AGAINST]->() RETURN r LIMIT 20;

//Cherche les relations PLAYED_AGAINST entre PLAYER et TEAM – Noeuds + Relation

MATCH (p:PLAYER) -[r:PLAYED_AGAINST]-> (t:TEAM) RETURN p, r, t;
```

8. Faire exécuter de petites requêtes

```
//Trouver tous les coéquipiers de LeBron :

MATCH (:PLAYER{name:"LeBron James"})-[:TEAMMATES]-(p)

RETURN p.name;

//Lister les joueurs coachés par Frank Vogel :

MATCH (:COACH{name:"Frank Vogel"})-[:COACHES]->(p)

RETURN p.name;

//Statistiques de LeBron contre Memphis :

MATCH (p:PLAYER{name:"LeBron James"})-[r:PLAYED_AGAINST]->(t:TEAM{name:"Memphis Grizzlies"})

RETURN r;
```

Attention à la sensibilité à la casse dans Neo4J

En gros, tout ce que « tu nommes » est sensible à la casse.

Élément	Exemple	Sensible à la casse ?	Explication
Label (type de nœud)	PLAYER, Team, coach	**Oui**	MATCH (n:PLAYER) ≠ MATCH (n:player)
Type de relation	PLAYS_FOR, plays_for, Plays_For	**Oui**	[:PLAYS_FOR] doit être écrit exactement comme lors de la création.
Nom de propriété	name, Name, NAME	**Oui**	n.name ≠ n.Name.
Valeur chaîne	"LeBron", "lebron"	**Oui**	Neo4j ne normalise pas les chaînes.
Alias de variable	p, P, player	**Oui**	Dans une même requête, p est différent de P.
Mots-clés Cypher	MATCH, match, MaTcH	**Non**	Les mots-clés ne sont **pas** sensibles à la casse.