

Référence rapide Neo4j / Cypher

Requêtes de base de données graphe, nœuds, relations, motifs

Fondamentaux Cypher

Structure de requête

MATCH	Trouver des motifs dans le graphe
WHERE	Filtrer les résultats
RETURN	Spécifier les colonnes de sortie
CREATE	Créer des nœuds et des relations
SET / REMOVE	Mettre à jour les propriétés et les étiquettes
DELETE / DETACH DELETE	Supprimer des nœuds et des relations

Exécution de requêtes

```
// Neo4j Browser: coller et exécuter avec Ctrl+Enter
// cypher-shell:
cypher-shell -u neo4j -p secret "MATCH (n) RETURN n LIMIT 5"
```

Nœuds et étiquettes

Syntaxe des nœuds

```
(n) // nœud anonyme
(p:Person) // nœud avec étiquette
(p:Person:Employee) // étiquettes multiples
(p:Person {name: "Alice", age: 30})
```

Opérations sur les étiquettes

```
SET n:Active // ajouter une étiquette
REMOVE n:Active // supprimer une étiquette
MATCH (n) RETURN labels(n) // lister les étiquettes
```

Contraintes et index

```
CREATE CONSTRAINT FOR (p:Person)
  REQUIRE p.email IS UNIQUE
CREATE INDEX FOR (p:Person) ON (p.name)
SHOW INDEXES
```

Relations

Syntaxe des relations

```
-[r]-> // orientée (sortante)
<-[r]- // orientée (entrante)
-[r]- // non orientée
-[:KNOWS]-> // relation typée
-[r:KNOWS {since: 2020}]-> // avec propriétés
```

Chemins à longueur variable

```
-[:KNOWS*2]-> // exactement 2 sauts
-[:KNOWS*1..3]-> // 1 à 3 sauts
-[:KNOWS*]-> // nombre quelconque de sauts
shortestPath((a)-[*]-(b)) // chemin le plus court
```

CREATE

Créer des nœuds

```
CREATE (p:Person {name: "Alice", age: 30})
CREATE (p:Person {name: "Bob"})
RETURN p
```

Créer des relations

```
MATCH (a:Person {name: "Alice"})
MATCH (b:Person {name: "Bob"})
CREATE (a)-[:KNOWS {since: 2020}]->(b)
```

MERGE (insertion ou mise à jour)

```
MERGE (p:Person {email: "alice@example.com"})
ON CREATE SET p.name = "Alice", p.created = date()
ON MATCH SET p.lastSeen = date()
```

MATCH

Motifs de base

```
MATCH (p:Person) RETURN p
MATCH (p:Person)-[:KNOWS]->(f) RETURN p, f
MATCH (a)-[r]->(b) RETURN type(r), a, b
```

OPTIONAL MATCH

```
// Retourne null pour les correspondances manquantes (comme LEFT JOIN)
MATCH (p:Person)
OPTIONAL MATCH (p)-[:OWNS]->(c:Car)
RETURN p.name, c.model
```

Compréhension de motifs

```
MATCH (p:Person)
RETURN p.name,
  [(p)-[:KNOWS]->(f) | f.name] AS friends
```

WHERE

Comparaison et logique

```
WHERE p.age > 25
WHERE p.age >= 18 AND p.active = true
WHERE p.name <> "Bob" OR p.role = "admin"
WHERE NOT (p)-[:BLOCKED]->()
```

Prédicats sur chaînes et listes

```
WHERE p.name STARTS WITH "Al"
WHERE p.name CONTAINS "ice"
WHERE p.name =~ "(?)alice.*" // regex
WHERE p.age IN [25, 30, 35]
```

Vérifications de null et d'existence

```
WHERE p.email IS NOT NULL
WHERE p.phone IS NULL
WHERE EXISTS { (p)-[:KNOWS]->(:Person) }
```

RETURN

Options de sortie

```
RETURN p.name AS name, p.age AS age
RETURN DISTINCT p.city
RETURN p, collect(f) AS friends
RETURN count(*) AS total
```

Tri et pagination

```
RETURN p.name ORDER BY p.age DESC
RETURN p SKIP 10 LIMIT 5
```

UNWIND

```
// Développer une liste en lignes
UNWIND [1, 2, 3] AS x RETURN x
UNWIND $names AS name
MERGE (p:Person {name: name})
```

MISE À JOUR ET SUPPRESSION

SET propriétés

```
MATCH (p:Person {name: "Alice"})
SET p.age = 31, p.updated = date()
SET p += {city: "NYC", active: true}
```

REMOVE

```
MATCH (p:Person {name: "Alice"})
REMOVE p.temp_field // supprimer une propriété
REMOVE p:Inactive // supprimer une étiquette
```

DELETE

```
MATCH (p:Person {name: "Bob"})
DETACH DELETE p // supprimer le nœud + toutes les relations
// DELETE p // échoue si le nœud a des relations
MATCH ()-[r:OLD_REL]->() DELETE r // supprimer une relation
```

Agrégation

Fonctions d'agrégation

count(x)	Nombre de valeurs non nulles
sum(x)	Somme des valeurs numériques
avg(x)	Moyenne des valeurs numériques
min(x) / max(x)	Valeur minimale / maximale
collect(x)	Agréger les valeurs dans une liste
percentileCont(x, 0.5)	Percentile continu

GROUP BY (implicite)

```
// Les colonnes non agrégées deviennent les clés de regroupement
MATCH (p:Person)-[:LIVES_IN]->(c:City)
RETURN c.name, count(p) AS population
ORDER BY population DESC
```

WITH (agrégation chaînée)

```
MATCH (p:Person)-[:KNOWS]->(f)
WITH p, count(f) AS friendCount
WHERE friendCount > 5
RETURN p.name, friendCount
```

Motifs courants

Trouver des amis communs

```
MATCH (a:Person {name: "Alice"})-[:KNOWS]->(m)-[:KNOWS]-(b:Person
{name: "Bob"})
RETURN m.name AS mutualFriend
```

Recommandation (amis d'amis)

```
MATCH (p:Person {name: "Alice"})-[:KNOWS*2]-(fof)
WHERE NOT (p)-[:KNOWS]-(fof) AND p <> fof
RETURN DISTINCT fof.name
```

Importer des données CSV

```
LOAD CSV WITH HEADERS FROM 'file:///people.csv' AS row
MERGE (p:Person {id: row.id})
SET p.name = row.name, p.age = toInteger(row.age)
```

Informations sur la base

```
CALL db.labels() // lister toutes les étiquettes
CALL db.relationshipTypes() // lister les types de relations
CALL db.schema.visualization()
```