

Neo4j / Cypher クイックリファレンス

グラフデータベースのクエリ、ノード、リレーションシップ、パターン

Cypher の基本

クエリ構造

MATCH	グラフ内のパターンを検索
WHERE	結果をフィルタリング
RETURN	出力列を指定
CREATE	ノードとリレーションシップを作成
SET / REMOVE	プロパティとラベルを更新
DELETE / DETACH DELETE	ノードとリレーションシップを削除

クエリの実行

```
// Neo4j Browser: Ctrl+Enterで貼り付けて実行
// cypher-shell:
cypher-shell -u neo4j -p secret "MATCH (n) RETURN n LIMIT 5"
```

ノードとラベル

ノードの構文

```
(n) // 匿名ノード
(p:Person) // ラベル付きノード
(p:Person:Employee) // 複数ラベル
(p:Person {name: "Alice", age: 30})
```

ラベル操作

```
SET n:Active // ラベルを追加
REMOVE n:Active // ラベルを削除
MATCH (n) RETURN labels(n) // ラベルの一覧表示
```

制約とインデックス

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

リレーションシップ

リレーションシップの構文

```
-[r]-> // 有向 (送信方向)
<-[r]- // 有向 (受信方向)
-[r]- // 無向
-[:KNOWS]-> // 型付きリレーションシップ
-[r:KNOWS {since: 2020}]-> // プロパティ付き
```

可変長パス

```
-[:KNOWS*2]-> // ちょうど2ホップ
-[:KNOWS*1..3]-> // 1~3ホップ
-[:KNOWS*]-> // 任意のホップ数
shortestPath((a)-[*]-(b)) // 最短パス
```

CREATE

ノードの作成

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

リレーションシップの作成

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

MERGE (アップサート)

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

MATCH

基本パターン

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

OPTIONAL MATCH

```
// 一致がない場合はnullを返す (LEFT JOINと同様)
MATCH (p:Person)
OPTIONAL MATCH (p)-[:OWNS]->(c:Car)
RETURN p.name, c.model
```

パターン内包表記

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

WHERE

比較と論理

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

文字列とリストの述語

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

Null と Existence のチェック

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

RETURN

出力オプション

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

並べ替えとページネーション

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

UNWIND

```
// リストを行に展開
UNWIND [1, 2, 3] AS x RETURN x
UNWIND $names AS name
MERGE (p:Person {name: name})
```

更新と削除

プロパティの設定

```
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 // プロパティを削除
REMOVE p:Inactive // ラベルを削除
```

DELETE

```
MATCH (p:Person {name: "Bob"})
DETACH DELETE p // ノードと全リレーションシップを削除
// DELETE p // リレーションシップがある場合は失敗
MATCH ()-[r:OLD_REL]->() DELETE r // リレーションシップを削除
```

集計

集計関数

count(x)	非 null 値の数
sum(x)	数値の合計
avg(x)	数値の平均
min(x) / max(x)	最小値/最大値
collect(x)	値をリストに集計
percentileCont(x, 0.5)	連続パーセンタイル

GROUP BY (暗黙的)

```
// 非集計列がグルーピングキーになる
MATCH (p:Person)-[:LIVES_IN]->(c:City)
RETURN c.name, count(p) AS population
ORDER BY population DESC
```

WITH (連鎖集計)

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

よくあるパターン

共通の友人を探す

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

レコメンデーション (友人の友人)

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

CSV データのインポート

```
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)
```

データベース情報

```
CALL db.labels() // 全ラベルの一覧
CALL db.relationshipTypes() // リレーションシップ型の一覧
CALL db.schema.visualization()
```