NEO4J / CYPHER 빠른 참조

그래프 데이터베이스 쿼리, 노드, 관계, 패턴

Cypher 기본

쿼리 구조

`MATCH`	그래프에서 패턴 탐색
`WHERE`	결과 필터링
`RETURN`	출력 컬럼 지정
`CREATE`	노드 및 관계 생성
`SET / REMOVE`	속성 및 레이블 업데이트
`DELETE / DETACH DELETE`	노드 및 관계 제거

쿼리 실행

```
// Neo4j Browser: paste and run with Ctrl+Enter
// cypher-shell:
cypher-shell -u neo4j -p secret "MATCH (n) RETURN n LIMIT 5"
```

노드 및 레이블

노드 문법

```
(n)                    // anonymous node
(p:Person)             // node with label
(p:Person:Employee)    // multiple labels
(p:Person {name: "Alice", age: 30})
```

레이블 작업

```
SET n:Active           // add label
REMOVE n:Active        // remove label
MATCH (n) RETURN labels(n)  // list labels
```

제약 및 인덱스

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

관계

관계 문법

```
-[r]->                 // directed (outgoing)
<-[r]-                 // directed (incoming)
-[r]-                  // undirected
-[:KNOWS]->            // typed relationship
-[r:KNOWS {since: 2020}]->  // with properties
```

가변 길이 경로

```
-[:KNOWS*2]->          // exactly 2 hops
-[:KNOWS*1..3]->       // 1 to 3 hops
-[:KNOWS*]->           // any number of hops
shortestPath((a)-[*]-(b))  // shortest path
```

CREATE

노드 생성

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

관계 생성

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

MERGE (Upsert)

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

MATCH

기본 패턴

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

OPTIONAL MATCH

```
// Returns null for missing matches (like LEFT JOIN)
MATCH (p:Person)
OPTIONAL MATCH (p)-[:OWNS]->(c:Car)
RETURN p.name, c.model
```

패턴 컴프리헨션

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

WHERE

비교 및 논리

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

문자열 및 목록 조건

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

Null 및 존재 확인

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

RETURN

출력 옵션

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

정렬 및 페이지네이션

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

UNWIND

```
// Expand a list into rows
UNWIND [1, 2, 3] AS x RETURN x
UNWIND $names AS name
MERGE (p:Person {name: name})
```

업데이트 및 삭제

SET 속성

```
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     // remove property
REMOVE p:Inactive       // remove label
```

DELETE

```
MATCH (p:Person {name: "Bob"})
DETACH DELETE p          // delete node + all rels
// DELETE p              // fails if node has rels
MATCH ()-[r:OLD_REL]->() DELETE r  // delete rel
```

집계

집계 함수

`count(x)`	null이 아닌 값의 수
`sum(x)`	숫자 값의 합
`avg(x)`	숫자 값의 평균
`min(x) / max(x)`	최솟값 / 최댓값
`collect(x)`	값을 목록으로 집계
`percentileCont(x, 0.5)`	연속 백분위수

GROUP BY (암묵적)

```
// Non-aggregated columns become grouping keys
MATCH (p:Person)-[:LIVES_IN]->(c:City)
RETURN c.name, count(p) AS population
ORDER BY population DESC
```

WITH (체이닝 집계)

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

주요 패턴

공통 친구 찾기

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

추천 (친구의 친구)

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

CSV 데이터 가져오기

```
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)
```

데이터베이스 정보

```
CALL db.labels()        // list all labels
CALL db.relationshipTypes()  // list rel types
CALL db.schema.visualization()
```