NEO4J / CYPHER 快速参考

图数据库查询、节点、关系、模式匹配

Cypher 基础

查询结构

`MATCH`	在图中查找模式
`WHERE`	过滤结果
`RETURN`	指定输出列
`CREATE`	创建节点和关系
`SET / REMOVE`	更新属性和标签
`DELETE / DETACH DELETE`	删除节点和关系

运行查询

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

节点与标签

节点语法

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

标签操作

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

约束与索引

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

关系

关系语法

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

可变长路径

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

CREATE

创建节点

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

创建关系

```
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

基本模式

```
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
```

模式推导式

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

WHERE

比较与逻辑

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

字符串与列表谓词

```
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 与存在性检查

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

RETURN

输出选项

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

排序与分页

```
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})
```

更新与删除

SET 属性

```
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
```

聚合

聚合函数

`count(x)`	非空值的数量
`sum(x)`	数值之和
`avg(x)`	数值平均值
`min(x) / max(x)`	最小值 / 最大值
`collect(x)`	将值聚合为列表
`percentileCont(x, 0.5)`	连续百分位数

隐式 GROUP BY

```
// 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（链式聚合）

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

常见模式

查找共同好友

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

推荐（好友的好友）

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

导入 CSV 数据

```
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)
```

数据库信息

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