

NEO4J / CYPHER QUICK REFERENCE

Graph database queries, nodes, relationships, patterns

Cypher Basics

Query Structure

```
MATCH Find patterns in the graph
WHERE Filter results
RETURN Specify output columns
CREATE Create nodes and relationships
SET / REMOVE Update properties and labels
DELETE / DETACH DELETE Remove nodes and relationships
```

Running Queries

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

Nodes & Labels

Node Syntax

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

Label Operations

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

Constraints & Indexes

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

Relationships

Relationship Syntax

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

Variable-Length Paths

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

CREATE

Create Nodes

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

Create Relationships

```
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

Basic Patterns

```
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
```

Pattern Comprehension

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

WHERE

Comparison & Logic

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

String & List Predicates

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

Null & Existence Checks

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

RETURN

Output Options

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

Ordering & Pagination

```
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})
```

UPDATE & DELETE

SET Properties

```
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
```

Aggregation

Aggregate Functions

```
count(x) Number of non-null values
sum(x) Sum of numeric values
avg(x) Average of numeric values
min(x) / max(x) Minimum / maximum value
collect(x) Aggregate values into a list
percentileCont(x, 0.5) Continuous percentile
```

GROUP BY (Implicit)

```
// 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 (Chained Aggregation)

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

Common Patterns

Find Mutual Friends

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

Recommendation (Friends of Friends)

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

Import CSV Data

```
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)
```

Database Info

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