# Referencia Rápida de Neo4j / Cypher

Consultas de grafos, nodos, relaciones, patrones

## Fundamentos de Cypher

### Estructura de la consulta

| | |
|---|---|
| **MATCH** | Encontrar patrones en el grafo |
| **WHERE** | Filtrar resultados |
| **RETURN** | Especificar columnas de salida |
| **CREATE** | Crear nodos y relaciones |
| **SET / REMOVE** | Actualizar propiedades y etiquetas |
| **DELETE / DETACH DELETE** | Eliminar nodos y relaciones |

### Ejecutar consultas

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

## Nodos y etiquetas

### Sintaxis de nodo

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

### Operaciones de etiqueta

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

### Restricciones e índices

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

## Relaciones

### Sintaxis de relación

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

### Rutas de longitud variable

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

## CREATE

### Crear nodos

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

### Crear relaciones

```
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

### Patrones básicos

```
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
```

### Comprensión de patrones

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

## WHERE

### Comparación y lógica

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

### Predicados de cadena y lista

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

### Verificaciones de nulo y existencia

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

## RETURN

### Opciones de salida

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

### Ordenar y paginar

```
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})
```

## ACTUALIZAR y ELIMINAR

### SET de propiedades

```
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
```

## Agregación

### Funciones de agregación

| | |
|---|---|
| **count(x)** | Número de valores no nulos |
| **sum(x)** | Suma de valores numéricos |
| **avg(x)** | Promedio de valores numéricos |
| **min(x) / max(x)** | Valor mínimo / máximo |
| **collect(x)** | Agregar valores en una lista |
| **percentileCont(x, 0.5)** | Percentil continuo |

### GROUP BY (implícito)

```
// 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 (agregación encadenada)

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

## Patrones comunes

### Encontrar amigos mutuos

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

### Recomendación (amigos de amigos)

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

### Importar datos 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)
```

### Información de la base de datos

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