Neo4j Cypher Complete Master Guide

Table of Contents

Relationship Symbols & Patterns

Basic Relationship Symbols

cypher

```cypher
// Direction Control
()-[]->()                    // Outgoing relationship
()<-[]-()                    // Incoming relationship
()-[]-()                     // Undirected (bidirectional)

// Typed Relationships
()-[:TYPE]->()               // Specific relationship type
()-[:TYPE1|:TYPE2]->()       // Multiple types
()-[:TYPE*]->()              // Variable length
```

Advanced Relationship Patterns

cypher

```cypher
// Quantified Path Patterns
```

```cypher
(a)-[:KNOWS]{1,3}->(b)       // 1 to 3 occurrences
(a)-[:KNOWS]{2}->(b)         // Exactly 2 occurrences
(a)-[:KNOWS]+->(b)           // One or more (Kleene plus)
(a)-[:KNOWS]*->(b)           // Zero or more (Kleene star)
(a)-[:KNOWS]?->(b)           // Zero or one (optional)

// Negative Patterns
WHERE NOT (a)-[:BLOCKS]->(b) // Exclusion patterns
WHERE NONE(r IN relationships(path) WHERE r.weight < 0)
```

Complex Path Expressions

cypher

```cypher
// Branching Patterns
(a)-[:A]->(b)-[:B]->(c)<-[:C]-(d)-[:D]->(e)

// Diamond Patterns
(a)-[:REL1]->(b)<-[:REL2]-(c)-[:REL3]->(d)

// Star Patterns
(center)-[:CONNECTS]->(node1),
(center)-[:CONNECTS]->(node2),
(center)-[:CONNECTS]->(node3)
```

Relationship Property Patterns

cypher

```cypher
// Property-based Relationships
()-[r {active: true, since: date()}]->()
()-[r WHERE r.weight > 0.5]->()
()-[r WHERE r.startDate < r.endDate]->()

// Temporal Relationships
()-[r:INTERACTION WHERE r.timestamp > datetime()-duration('P7D')]->()
```

Advanced Node Operations

Dynamic Node Creation

cypher

```cypher
// Batch Creation
UNWIND range(1, 100) AS id
CREATE (n:Node {id: id, created: timestamp()})

// Conditional Creation
CALL apoc.merge.node(
  ['Person'],
  {ssn: $ssn},
  {name: $name, age: $age}
) YIELD node
// Procedures return results, but you need YIELD to access them:
```

Memory and Performance Effects

- **YIELD filters columns early** - reduces memory usage
- **Without YIELD** - you might get unnecessary data
- **With WHERE after YIELD** - efficient filtering

YIELD is essential for:

- Accessing procedure results
- Renaming output columns
- Filtering procedure output
- Integrating procedures into larger queries
- Memory efficiency by selecting only needed columns

Bottom line: Think of YIELD as the RETURN clause for procedures - it defines wh at data comes out of the procedure and makes it available to the rest of your q uery.

```cypher
// Pattern-based Creation
MATCH (a:Person), (b:Company)
CREATE (a)-[r:WORKS_AT {since: date()}]->(b)
```

Advanced Property Management

```cypher
// Map Projections
MATCH (p:Person)
RETURN p {.name, .age, .address} AS personInfo

// Dynamic Properties
WITH {name: 'Alice', age: 30, city: 'Boston'} AS data
CREATE (p:Person) SET p += data

// JSON Integration
CALL apoc.load.json('file:///data.json') YIELD value
CREATE (n:Data) SET n = value
```

Node Functions Deep Dive

```cypher
// Advanced ID Management
elementId(n)                 // Modern element ID
id(n)                        // Legacy internal ID

// Label Operations
LABELS(n)                    // All labels
n:Label                      // Label check in WHERE
CASE WHEN n:Important THEN true ELSE false END

// Property Functions
properties(n)                // All as map
n[$dynamicKey]               // Dynamic property access
apoc.map.removeKeys(n, ['temp', 'cache']) // Clean properties
```

Relationship Mastery

Advanced Relationship Creation

```cypher
// Bulk Relationship Creation
MATCH (p:Person), (c:Company)
WHERE p.companyId = c.id
CREATE (p)-[r:EMPLOYED_BY {
  since: p.startDate,
  department: p.dept
}]->(c)

// Conditional Relationships
MATCH (a:User), (b:User)
WHERE a.trustScore > 0.8 AND b.trustScore > 0.8
MERGE (a)-[r:TRUSTS]->(b)
ON CREATE SET r.strength = 0.5, r.created = datetime()
ON MATCH SET r.strength = r.strength + 0.1

// Temporal Relationships
CREATE (a)-[r:INTERACTED {
  timestamp: datetime(),
  duration: duration('PT30M'),
  intensity: 0.75
}]->(b)
```

Relationship Analytics

```cypher
// Relationship Metrics
MATCH (a)-[r]->(b)
RETURN
  TYPE(r) AS relationshipType,
  COUNT(r) AS frequency,
  AVG(r.weight) AS avgWeight,
  MIN(r.weight) AS minWeight,
  MAX(r.weight) AS maxWeight,
  COLLECT(DISTINCT LABELS(a)) AS sourceLabels,
  COLLECT(DISTINCT LABELS(b)) AS targetLabels

// Network Analysis
MATCH (a)-[r:INTERACTS]->(b)
```

```cypher
WITH a, b, COUNT(r) AS interactionCount
WHERE interactionCount > 5
CREATE (a)-[s:STRONG_TIE {weight: interactionCount}]->(b)
```

Complex Relationship Patterns

cypher

```cypher
// Multi-hop with Conditions
MATCH path = (a:Person)-[rels:KNOWS*2..5]->(b:Person)
WHERE ALL(r IN rels WHERE r.since > date('2020-01-01'))
  AND a.city = b.city
  AND NONE(n IN nodes(path) WHERE n:Blacklisted)
RETURN path, length(path) AS distance

// Recursive Patterns
MATCH (root:Category {name: 'Electronics'})
CALL {
  WITH root
  MATCH path = (root)-[:SUBCATEGORY*0..]->(subcat)
  RETURN subcat
}
RETURN subcat.name, length(path) AS depth
```

Query Clauses Deep Dive

Advanced MATCH Patterns

cypher

```cypher
// Optional Match with Multiple Patterns
MATCH (p:Person)
OPTIONAL MATCH (p)-[:WORKS_AT]->(company)
OPTIONAL MATCH (p)-[:LIVES_IN]->(city)<-[:LOCATED_IN]-(company)
RETURN p.name, company.name, city.name

// Shortest Path with Filters
```

```cypher
MATCH path = shortestPath(
  (a:Person {name: 'Alice'})-[:KNOWS|WORKS_WITH*1..6]-(b:Person {name: 'Bob'})
)
WHERE ALL(node IN nodes(path) WHERE node.age > 18)
RETURN path, length(path) AS degrees

// All Shortest Paths
MATCH paths = allShortestPaths(
  (a:Location)-[:CONNECTED*]-(b:Location)
)
WHERE a.region = b.region
RETURN paths, length(paths) AS hopCount
```

WITH Clause Advanced Usage

cypher

```cypher
// Pipeline Aggregation
MATCH (p:Person)-[:FRIENDS]->(friend)
WITH p, COLLECT(friend) AS friends
WHERE SIZE(friends) > 10
WITH p, friends,
     [f IN friends WHERE f.age > 30] AS olderFriends
RETURN p.name,
       SIZE(friends) AS totalFriends,
       SIZE(olderFriends) AS matureFriends

// Window Functions Simulation
MATCH (p:Person)
WITH p ORDER BY p.salary DESC
WITH COLLECT(p) AS people
UNWIND RANGE(0, SIZE(people)-1) AS idx
WITH people[idx] AS person, idx + 1 AS rank
RETURN person.name, person.salary, rank
```

CALL Subqueries Mastery

cypher

```cypher
// Correlated Subqueries with Parameters
```

```cypher
MATCH (department:Department)
CALL {
  WITH department
  MATCH (department)<-[:IN_DEPT]-(employee:Employee)
  RETURN AVG(employee.salary) AS avgSalary,
         COUNT(employee) AS headcount
}
RETURN department.name, avgSalary, headcount

// Union in Subqueries
CALL {
  MATCH (p:Person)-[:CREATED]->(post:Post)
  RETURN p AS user, post AS content, 'post' AS type
  UNION
  MATCH (p:Person)-[:LIKED]->(post:Post)
  RETURN p AS user, post AS content, 'like' AS type
}
RETURN user.name, content.title, type
```

Functions & Aggregation

Advanced String Functions

cypher

```cypher
// Text Processing
RETURN
  apoc.text.clean('  Hello  World  ') AS cleaned,
  apoc.text.join(['Hello', 'World'], ' ') AS joined,
  apoc.text.slug('Hello World!') AS slug,
  apoc.text.base64Encode('secret') AS encoded

// Regular Expressions
WHERE n.email =~ '[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}'
RETURN apoc.text.regexGroups('Price: $100', '\$(\d+)') AS matches
```

Temporal Functions Deep Dive

cypher

```cypher
// Date/Time Operations
RETURN
  datetime() AS now,
  datetime().year AS currentYear,
  date() AS today,
  time() AS currentTime,
  datetime() + duration('P1DT2H') AS tomorrow,
  duration.between(startDate, endDate) AS difference

// Business Date Calculations
RETURN
  apoc.date.add(timestamp(), 'd', 5, 'BUSINESS') AS in5BusinessDays,
  apoc.date.parse('2024-12-25', 's', 'yyyy-MM-dd') AS christmas
```

Spatial Operations

cypher

```cypher
// Geographic Calculations
WITH
  point({latitude: 40.7128, longitude: -74.0060}) AS nyc,
  point({latitude: 34.0522, longitude: -118.2437}) AS la
RETURN
  distance(nyc, la) AS distanceInMeters,
  point.distance(nyc, la) AS alternative

// Spatial Indexing
CREATE POINT INDEX location_index FOR (p:Place) ON (p.location)
MATCH (p:Place)
WHERE point.distance(p.location, point({latitude: 40.7, longitude: -74.0})) < 1
0000
RETURN p.name
```

Advanced Aggregation

cypher

```cypher
// Statistical Aggregation
```

```cypher
MATCH (p:Person)
RETURN
  COUNT(p) AS total,
  AVG(p.age) AS averageAge,
  PERCENTILE_DISC(0.5, p.age) AS medianAge,
  STDEV(p.salary) AS salaryStdDev,
  COLLECT(p.age) AS allAges,
  apoc.coll.percentile(allAges, 0.95) AS percentile95

// Rolling Aggregates
MATCH (day:Day)-[:NEXT]->(next:Day)
WITH day, next ORDER BY day.date
WITH COLLECT({date: day.date, sales: day.sales}) AS days
UNWIND RANGE(0, SIZE(days)-7) AS start
WITH days[start..start+7] AS week
RETURN
  week[0].date AS weekStart,
  SUM(d.sales IN week) AS weeklySales,
  AVG(d.sales IN week) AS avgDailySales
```

Performance & Optimization

Query Optimization Techniques

cypher

```cypher
// Index Hints
MATCH (p:Person) USING INDEX p:Person(name)
WHERE p.name = 'Alice'
RETURN p

// Scan Hints
MATCH (p:Person) USING SCAN p:Person
WHERE p.name STARTS WITH 'A'
RETURN p
```

```cypher
// Join Hints
MATCH (a:Person), (b:Person)
USING JOIN ON a
WHERE a.managerId = b.id
RETURN a, b
```

Advanced Profiling

```cypher
// Detailed Query Analysis
PROFILE
MATCH (p:Person)-[:FRIENDS_WITH*2..3]->(fof:Person)
WHERE p.age > 25 AND fof.age > 25
RETURN p.name, COLLECT(DISTINCT fof.name) AS network
```

Memory Management

```cypher
// Batch Processing
CALL apoc.periodic.iterate(
  "MATCH (p:Person) RETURN p",
  "SET p.processed = true",
  {batchSize: 1000, parallel: true}
)

// Memory-efficient Aggregation
MATCH (p:Person)
WITH p ORDER BY p.id
WITH COLLECT(p) AS batch, COUNT(*) AS total
UNWIND batch AS person
RETURN person.name, total
```

Advanced Patterns & Algorithms

Graph Algorithms

cypher

```cypher
// Centrality Algorithms
CALL gds.pageRank.stream({
  nodeQuery: 'MATCH (p:Person) RETURN id(p) AS id',
  relationshipQuery: 'MATCH (p1:Person)-[:FOLLOWS]->(p2:Person) RETURN id(p1) A
S source, id(p2) AS target',
  maxIterations: 20,
  dampingFactor: 0.85
}) YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS person, score
ORDER BY score DESC

// Community Detection
CALL gds.louvain.stream({
  nodeQuery: 'MATCH (p:Person) RETURN id(p) AS id',
  relationshipQuery: 'MATCH (p1:Person)-[:INTERACTS]->(p2:Person) RETURN id(p1)
AS source, id(p2) AS target',
  relationshipWeightProperty: 'weight'
}) YIELD nodeId, communityId
RETURN communityId, COLLECT(gds.util.asNode(nodeId).name) AS members
```

Path Finding Algorithms

cypher

```cypher
// Dijkstra Shortest Path
MATCH (start:Location {name: 'A'}), (end:Location {name: 'F'})
CALL gds.shortestPath.dijkstra.stream({
  nodeQuery: 'MATCH (l:Location) RETURN id(l) AS id',
  relationshipQuery: 'MATCH (l1:Location)-[r:ROAD]->(l2:Location) RETURN id(l1)
AS source, id(l2) AS target, r.distance AS cost',
  startNode: start,
  endNode: end,
  relationshipWeightProperty: 'cost'
}) YIELD path
RETURN path
```

Machine Learning Integration

```cypher
// Node Classification
CALL gds.alpha.ml.nodeClassification.train({
  nodeProjection: 'Person',
  relationshipProjection: 'KNOWS',
  featureProperties: ['age', 'salary', 'education'],
  targetProperty: 'profession',
  modelName: 'person-profession'
})

// Link Prediction
CALL gds.alpha.ml.linkPrediction.train({
  nodeProjection: 'Person',
  relationshipProjection: 'KNOWS',
  modelName: 'friendship-predictor'
})
```

Administration & Security

User & Role Management

```cypher
// Advanced User Management
CREATE USER alice SET PASSWORD $password CHANGE NOT REQUIRED
SET PASSWORD CHANGE REQUIRED
ALTER USER alice SET STATUS SUSPENDED

// Role-based Access Control
CREATE ROLE data_scientist
GRANT MATCH {*} ON GRAPH neo4j NODES Person TO data_scientist
GRANT TRAVERSE ON GRAPH neo4j RELATIONSHIPS KNOWS TO data_scientist
DENY READ {salary} ON GRAPH neo4j NODES Person TO data_scientist
```

Database Administration

```cypher
// Multi-database Management
CREATE DATABASE analytics
CREATE COMPOSITE DATABASE federated
SHOW DATABASES
:USE analytics

// Backup and Restore
CALL apoc.export.cypher.all('backup.cypher', {})
CALL apoc.import.cypher('backup.cypher', {})
```

Monitoring & Metrics

```cypher
// Query Performance Monitoring
SHOW TRANSACTIONS
SHOW QUERIES
KILL QUERIES 'query-id'

// Database Metrics
CALL db.stats.retrieve('GRAPH COUNTS')
CALL db.stats.retrieve('TOKENS')
```

APOC Procedures

Data Import/Export

```cypher
// JSON Integration
CALL apoc.load.json('https://api.example.com/data') YIELD value
CREATE (n:ExternalData) SET n = value

// CSV Processing
```

```cypher
CALL apoc.load.csv('file:///data.csv', {header: true}) YIELD map
CREATE (n:Record) SET n += map

// Excel Integration
CALL apoc.load.xls('file:///data.xlsx', 'Sheet1', {header: true}) YIELD map
```

Advanced Data Processing

cypher

```cypher
// Data Cleaning
CALL apoc.clean('  HELLO  world 123  ') // "hello world 123"

// Hashing
RETURN apoc.util.sha1([1,2,3]) // "6f571b012e7..."

// Cryptography
RETURN apoc.util.md5('secret')
```

Best Practices & Patterns

Enterprise Patterns

cypher

```cypher
// Temporal Data Pattern
CREATE (e:Event {
  id: apoc.create.uuid(),
  timestamp: datetime(),
  type: 'USER_LOGIN'
})

// Versioning Pattern
CREATE (v1:Document {id: 'doc1', version: 1, content: 'v1'})
CREATE (v2:Document {id: 'doc1', version: 2, content: 'v2'})
CREATE (v1)-[:NEXT_VERSION]->(v2)
```

```cypher
// Audit Trail Pattern
CREATE (action:AuditAction {
  timestamp: datetime(),
  user: 'alice',
  action: 'CREATE',
  resource: 'Person/123'
})
```

Performance Patterns

```cypher
// Eager Loading Prevention
MATCH (p:Person)
CALL {
  WITH p
  MATCH (p)-[:FRIENDS]->(f)
  RETURN COLLECT(f.name) AS friends
}
RETURN p.name, friends

// Memory Management
CALL apoc.periodic.commit(
  "MATCH (p:Person) WHERE p.processed IS NULL
   WITH p LIMIT $limit
   SET p.processed = true
   RETURN COUNT(p)",
  {limit: 10000}
)
```

This comprehensive guide covers Neo4j Cypher from basic symbols to enterprise-level patterns, including performance optimization, advanced algorithms, and administration techniques.