Cypher is the declarative query language for Neo4j, the world's leading graph database.

Key principles and capabilities of Cypher are as follows:

- **Cypher matches patterns of nodes and relationship in the graph, to extract information or modify the data.**
- **Cypher has the concept of identifiers which denote named, bound elements and parameters.**
- **Cypher can create, update, and remove nodes, relationships, labels, and properties.**
- **Cypher manages indexes and constraints.**

You can try Cypher snippets live in the Neo4j Console at the [Neo4j Console](https://neo4j.com/) or read the full Cypher documentation in the Neo4j Manual. For live graph models using Cypher check out [GraphSpace](https://graphspace.io/).

The Cypher Refcard is also available in PDF format.

![Neo4j Cypher Refcard 2.3.12](https://github.com/neo4j/cypher-refcard/raw/master/doc/neo4j-cypher-refcard.png)

**Note:** (value) denotes either literals, for ad hoc Cypher queries, or parameters, which is the best practice for applications. Neo4j properties can be strings, numbers, booleans or arrays thereof. Cypher also supports maps and collections.

### Syntax

#### Read Query Structure

**MATCH**

```cypher
MATCH (a)-[r]-(b) RETURN a, r, b
```

**CREATE**

```cypher
CREATE (a) RETURN a
```

#### Write-Only Query Structure

**CREATE (a) RETURN a**

Use a predicate to filter. Note that what is always part of a `MATCH` clause is the `RETURN` clause. Putting it after a different clause in a `WHERE` will alter what it does.

**MATCH (a)-[r]-(b) RETURN a, r, b WHERE a.name = 'Alice'**

**WHERE**

```cypher
WHERE a.name = 'Alice'
```

**DELETE**

```cypher
DELETE (a)
```

#### UPSERT (a) RETURN a

Add a node to a collection.

#### UPDATE**

```cypher
UPDATE (a)
```

#### SET**

```cypher
SET a.name = 'Alice'
```

#### MATCH (a)-[r]-(b) REMOVE r

Remove a relationship.

#### MATCH (a)-[r]-(b) REMOVE a FROM r

Remove a node and all relationships connected to it.

#### MATCH (a)-[r]-(b) REMOVE b FROM r

Remove all nodes and relationships from the database.

#### DELETE**

```cypher
DELETE (a)
```

#### REMOVE**

```cypher
REMOVE a
```

#### REMOVE**

```cypher
REMOVE a
```

#### Execute a matching operation for each relationship of a path.

**MATCH (a)-[r]-(b) WHERE a.name = 'Alice' RETURN r**

#### Execute a mutating operation for each element in a collection.

**MATCH (a)-[r]-(b) WHERE a.name = 'Alice' RETURN r**

#### Mathematical Comparison

- `<` 
- `<=`
- `>` 
- `>=`
- `=` 
- `<>`
- `?=`
- `!` 
- `!=`
- `= ?`

**Logical Expression**

- `NOT`
- `AND`
- `OR`

**String**

- `LIKE`
- `IN`
- `NOT LIKE`
- `NOT IN`
- `IS`
- `IS NOT`
- `CONTAINS`

**Collection**

- `COLLECT`
- `GROUP BY`
- `ORDER BY`
- `LIMIT`
- `OFFSET`
- `OFFSET`
- `UNION`
- `INTERSECT`
- `MINUS`
- `TOP X`
- ` DISTINCT`

**Array**

- `SIZE`
- `INDEX`
- `INDEXES`
- `APPLY`

**Map**

- `MAP`
Cypher is the declarative query language for Neo4j, the world's leading graph database.

Key principles and capabilities of Cypher are as follows:

- Cypher matches patterns of nodes and relationship in the graph, to extract information or modify the data.
- Cypher has the concept of identifiers which denote named, boxed elements and parameters.
- Cypher can create, update, and remove nodes, relationships, labels, and properties.
- Cypher manages indexes and constraints.

You can try Cypher snippets live in the Neo4j Console at console.neo4j.org or read full Cypher documentation in the Neo4j Manual. For live graph models using Cypher check out Cytopia.

The Cypher Reference is also available in PDF format.

Note (value): denotes either literals, for ad hoc Cypher queries; or parameters, which is the best practice for applications. Neo4j properties can be strings, numbers, booleans or arrays thereof. Cypher also supports maps and collections.

### Syntax

#### Read Query Structure

| MATCH WHERE | MATCH (a:Person) - [r:KNOWS] - (b:Person) WHERE a.name = "Alice" |
| MATCH WHERE | MATCH (a:Person) - [r:KNOWS] - (b:Person) WHERE a.name = "Alice" |

Node patterns can contain labels and properties.

Any pattern can be used in MATCH

MATCH WHERE a.id = 5

Assign a path to p

MATCH WHERE p = (a)-[:KNOWS]->(b)

Optional pattern, w may be used for missing parts.

MATCH WHERE w:Person - [r:KNOWS] - w:Person

Force the planner to use a label scan to solve the query (for manual performance tuning).

WHERE p:Person - [r:KNOWS] - p:Person

#### Write-Only Query Structure

| CREATE | CREATE (p:Person) CREATE (p:Person) |
| CREATE | CREATE (p:Person) CREATE (p:Person) |

Create a node with the given properties.

CREATE (p {name: "Alice", age: 18})

Create a relationship with the given type and direction; bind an identifier to it.

CREATE (p)-[:KNOWS]->(q)

Delete a node and a relationship.

DELETE p, (q:Person) - [r:KNOWS] - p

Delete all nodes and relationships from the database.

DELETE (q)

#### Match

| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |
| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |

Get all properties. This will remove any existing properties.

MATCH WHERE a = (p)

Add any properties, while keeping existing ones.

MATCH WHERE a = (p) RETURN a

Add a label to a node.

MATCH WHERE a = (p) RETURN a

#### Create

| CREATE | CREATE (p:Person) |
| CREATE | CREATE (p:Person) |

Create a node with the given properties.

CREATE (p {name: "Alice", age: 18})

Create a relationship with the type, createDirection and properties.

CREATE (p)-[:KNOWS]->(q {name: "Bob", age: 20})

Use a predicate to filter. Note that isIn is always part of a MATCH, optIn is t | false clause. Putting it after a different clause in a query will alter what it does.

WHERE p:Person - [r:KNOWS] - q:Person

#### Update

| CREATE | CREATE (p:Person) |
| CREATE | CREATE (p:Person) |

Delete a node and all relationships connected to it.

DELETE p, (q)-[:KNOWS]-(p)

Delete all nodes and relationships from the database.

DELETE q

Remove a label from a node.

MATCH WHERE a = (p) REMOVE a:label

Remove a property.

MATCH WHERE a = (p) REMOVE a.property

Execute a mutating operation for each relationship of a path.

MATCH WHERE a = (p) CREATE (a)-[:KNOWS]->(q)

Execute a mutating operation for each element in a collection.

MATCH WHERE a = (p) CREATE (a)

### Mathematical

| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |
| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |

Add labels to a node.

MATCH WHERE a = (p) RETURN a

Add a label to a node.

MATCH WHERE a = (p) RETURN a

### String

| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |
| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |

Create a node with labels and property.

MATCH WHERE a = (p) RETURN a

Create a node with unique node_id and label and properties.

MATCH WHERE a = (p) RETURN a

### Labels

| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |
| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |

Get the current labels of a node.

MATCH WHERE a = (p) RETURN a

Remove label from node.

MATCH WHERE a = (p) REMOVE a:label

### Operations

| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |
| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |

Add label to node.

MATCH WHERE a = (p) RETURN a

### Collections

| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |
| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |

Return the number of matching rows. See Aggregation for more information.

MATCH WHERE a = (p) RETURN a

#### Paths

| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |
| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |

### Relationships

| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |
| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |

Return the relationship count of a node.

MATCH WHERE a = (p) RETURN a

Return the relationship count of a node.

MATCH WHERE a = (p) RETURN a

### Aggregation

| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |
| MATCH WHERE | MATCH (a:Person) WHERE a.name = "Alice" |

### Syntax and examples

- Use variables instead of literals where possible. This allows you to be sure your queries are useful instead of having to parse and build new execution plans.
- Always set an upper limit for your variable length collections as the query planner needs to have a query graph for each all nodes in a graph by mistake.
- Return the data you need. Avoid returning whole nodes and relationships — instead, pick the data you need and return only that.
- Use EXPLAIN to analyze the performance of your queries. See Query Tuning for more information.