Neo4j Cypher Cheatsheet

A comprehensive cheat sheet for Neo4j's Cypher query language, covering syntax, common operations, and best practices for graph database interactions.

Cypher Basics

Nodes and Relationships

Nodes represent entities in the graph. Relationships define connections between nodes.

Nodes: `(node_name:Label {property1: value1, property2: value2})`
Relationships: `-[relationship_name:RELATIONSHIP_TYPE {property1: value1}]->`

Example of Node creation:

```
CREATE (p:Person {name: 'Alice', age: 30})
```

Example of Relationship creation:

```
MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'})
CREATE (a)-[r:KNOWS]->(b)
```

Basic Syntax

`MATCH`	Used to find nodes and relationships in the graph based on a pattern.
`CREATE`	Used to create new nodes and relationships in the graph.
`SET`	Used to update properties of nodes and relationships.
`DELETE`	Used to delete nodes and relationships.
`REMOVE`	Used to remove properties or labels from nodes and relationships.
`RETURN`	Specifies what data should be returned by the query.

Common Clauses

`WHERE` : Filters the results based on specified conditions.
`ORDER BY` : Sorts the results based on specified properties.
`LIMIT` : Limits the number of results returned.
`SKIP` : Skips a specified number of results.

Example:

```
MATCH (n:Person) WHERE n.age > 25 RETURN n
ORDER BY n.name LIMIT 10 SKIP 5
```

Pattern Matching

Basic Pattern Matching

Matching Nodes: `MATCH (n:Label)`
Matching Relationships: `MATCH (n)-[r:REL_TYPE]->(m)`
Matching Paths: `MATCH p=(n)-[r*]->(m)` (variable length paths)

Examples:

```
MATCH (a:Person)-[:KNOWS]->(b:Person) RETURN a, b
MATCH (a {name: 'Alice'})-[:KNOWS]->(b) RETURN b
```

Variable Length Relationships

`-[r*n]->`	Match relationships of exactly length `n`.
`-[r*n..m]->`	Match relationships of length between `n` and `m`.
`-[r*n..]->`	Match relationships of minimum length `n`.
`-[r*..m]->`	Match relationships of maximum length `m`.
`-[r*]->`	Match relationships of any length (including zero).

Directional Relationships

Directed: `(a)-[:REL]->(b)`
Undirected: `(a)-[:REL]-(b)`
Direction doesn't matter: `(a)<-[:REL]->(b)`

Example:

```
MATCH (a:Person)-[:FRIEND_OF]->(b:Person)
RETURN a, b
```

Data Manipulation

Creating Nodes and Relationships

Creating a Node: `CREATE (n:Label {properties})`
Creating a Relationship: `CREATE (a)-[r:REL_TYPE {properties}]->(b)`

Example:

```
CREATE (c:City {name: 'New York', country: 'USA'})
MATCH (p:Person {name: 'Alice'}), (c:City {name: 'New York'})
CREATE (p)-[:LIVES_IN]->(c)
```

Updating Data

`SET n.property = value`	Updates or creates a property on a node or relationship.
`SET n = {properties}`	Replaces all properties on a node or relationship.
`REMOVE n.property`	Removes a specific property from a node or relationship.
`REMOVE n:Label`	Removes a label from a node.

Deleting Data

`DELETE n` : Deletes a node.
`DELETE r` : Deletes a relationship.
Note: You must first delete relationships connected to a node before deleting the node itself, or use `DETACH DELETE n`.

Example:

```
MATCH (n:Person {name: 'Alice'}) DETACH DELETE n
```

Advanced Cypher

Aggregations

Cypher supports aggregation functions like `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`, `COLLECT`.
These are typically used with the `WITH` clause to group and aggregate data.

Example:

```
MATCH (n:Person)-[:FRIEND_OF]->(f)
WITH n, count(f) AS friendCount
RETURN n.name, friendCount ORDER BY friendCount DESC
```

List Comprehension

`[x IN list WHERE condition | expression]`	Creates a new list based on an existing list, filtering and transforming elements.

Example:

```
MATCH (p:Person)
RETURN p.name, [friend IN [(p)-[:FRIEND_OF]->(f) | f.name] WHERE friend IS NOT NULL] AS friends
```

Procedures and Functions

Neo4j has built-in procedures and functions, and you can also create your own.
`CALL db.indexes()` : Lists all indexes.
`RETURN toInteger('42')` : Converts a value to an integer.

Example:

```
CALL db.indexes()
YIELD name, type
RETURN name, type
```