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 relationships in the graph, to extract information or modify the data.
- Cypher has the concept of variables which do not need to be bound, 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 console.neo4j.io or read the full Cypher documentation in the Neo4j Developer Manual. For free graph models using Cypher check out GraphiQL.

The Cypher Reference is also available in PDF format.

Note: (name) denotes either literals or maps, used for set or hop queries. The use of parameters is recommended in applications, and are denoted by $(name)$.

Neo4j properties can be strung, numbers, booleans or arrays thereof. Cypher also supports maps and lists.

Syntax

 RAW TEXT START

**READ**

- Deactivate a node and a relationship.
- Nothing and all relationships connected to it.
- Match nodes and all relationships connected to it.
- Deactivate a node.
- Match nodes and all relationships connected to it.

**CREATE**

- Create a node and a relationship.
- Nothing and all relationships connected to it.
- Match nodes and all relationships connected to it.
- Deactivate a node.
- Match nodes and all relationships connected to it.

**MATCH**

- Node patterns can contain labels and properties.
- Any pattern can be used in select.
- Patterns with node properties.
- Design a path via.

**OPTIONAL**

- Optional pattern will be used for missing parts.

**RETURN**

- Deactivate a node and a relationship.
- Nothing and all relationships connected to it.
- Match nodes and all relationships connected to it.
- Deactivate a node.
- Match nodes and all relationships connected to it.

**USING**

- The syntax is similar to return. It separates query parts explicitly, allowing you to declare which variables to carry over to the next part.
- The number of matching rows. See Aggregating Functions for more.

**WHERE**

- Returns the distinct union of all query results.
- Involves sorting and naming here to match.

**ORDER**

- Returns the union of all query results, including duplicated rows.
- Creates a node with the given properties.

**OPTIONAL**

- Create a node with the given properties.
- Create indexes on the given properties.
- List all indexes used in the database.
- List all indexes used in the database.
- List all unique indexes in the database.
- List all indexes used in the database.

**SET**

- Set a property on a node.
- Add and remove properties, while keeping existing ones.

**REMOVE**

- Removes a node or a relationship.

**START**

- Removes a node from a list.

**PROPERTY**

- An index can automatically be used for the list checks.

**DROP**

- Drop the index on the label and property.

**INDEX**

- Indexes can be enforced when Cypher uses a suboptimal index, or more than one index should be created.

**DATA**

- Stores additional information about the data, or values of all variables.

**USING**

- Use alias for result column name.

**RETURN**

- Return original records.
- Return the result.
- Return the result and sort.
- Return the result and sort.

**MATCH**

- Skip a number of results.
- Limit the number of results.
- Skip results at the top and limit the number of results.

**WHERE**

- Examine a matching relationship for each operation in a relationship path.
- Count the paths matching the pattern.

**ORDER**

- The number of matching rows. See Aggregating Functions for more.

**RETURN**

- Deactivate a node and a relationship.
- Nothing and all relationships connected to it.
- Match nodes and all relationships connected to it.
- Deactivate a node.
- Match nodes and all relationships connected to it.

**USING**

- The syntax is similar to return. It separates query parts explicitly, allowing you to declare which variables to carry over to the next part.
- The number of matching rows. See Aggregating Functions for more.

**WHERE**

- Returns the distinct union of all query results.
- Involves sorting and naming here to match.

**ORDER**

- Returns the union of all query results, including duplicated rows.
- Creates a node with the given properties.

**OPTIONAL**

- Create a node with the given properties.
- Create indexes on the given properties.
- List all indexes used in the database.
- List all indexes used in the database.
- List all unique indexes in the database.
- List all indexes used in the database.

**SET**

- Set a property on a node.
- Add and remove properties, while keeping existing ones.

**REMOVE**

- Removes a node or a relationship.

**START**

- Removes a node from a list.

**PROPERTY**

- An index can automatically be used for the list checks.

**DROP**

- Drop the index on the label and property.

**INDEX**

- Indexes can be enforced when Cypher uses a suboptimal index, or more than one index should be created.

**DATA**

- Stores additional information about the data, or values of all variables.
Cypher is the declarative query language for Neo4j, the world's leading graph database.

Key principles and capabilities of Cypher are as follows:

1. Cypher evaluates patterns and relationships in the graph, to extract information or modify the data.
2. Cypher has the concept of variables which can contain node, label, element, or parameter. Variables can be used in pattern matching and as arguments in functions.
3. Cypher supports option, and remove nodes, relationships, labels, and properties.
4. Cypher manages indexes and constraints. You can try to synthesize it into the Neo4j CQL (Cypher) language.

The Cypher Refcard is also available as a PDF document.

Note: (x) denotes either literals or maps, used for set hoc queries. The usage of parameters is recommended, and desired by the tool. Neo4j properties can be strings, numbers, booleans or arrays. Cypher also supports maps and lists.

**Syntax**

**Read Query Structure**

- MATCH (n)
- MATCH (n:Label)
- MATCH (n:Label)<-[:Type]-(m)
- MATCH (n)-[:Type]->(m)
- MATCH (n)<-[r:Type]-(m)
- MATCH (n:Label)<-[r:Type]-(m:Label)

**Set all properties.** This will remove any existing property that doesn't exist yields a null value.

**Write-Only Query Structure**

- CREATE (n:Label {property: value})
- CREATE (n:Label)<-[:Type]-(m)
- CREATE (n:Label)-[:Type]->(m)
- CREATE (n)-[:Type]->(m)
- CREATE (n:Label)<-[r:Type]-(m:Label)

**Write-Query Structure**

- MATCH (n)
- MATCH (n:Label)
- MATCH (n:Label)<-[:Type]-(m)
- MATCH (n)-[:Type]->(m)
- MATCH (n)<-[r:Type]-(m)
- MATCH (n:Label)<-[r:Type]-(m:Label)

**General**

- The pattern is similar to Select. It separates query parts explicitly, allowing you to declare which variables to carry over to the next part.
- The number of matching rows. See Aggregating Functions for more.

**Declarations**

- Create a node and its properties.
- Create a node and a relationship.
- Create a node and all relationships connected to it.

**Conditions**

- Use the value of all variables.
- Use also for result column name.
- Return next result.
- Skip the result.

**Relationships**

- Return all relationship properties.
- Return all relationship properties excluding.
- Return all relationship properties.

**Syntactic Structure**

- The parentheses are used for numbers, booleans or arrays.
- The parentheses are used for numbers, booleans or arrays.
- The parentheses are used for numbers, booleans or arrays.

**Aggregating Functions**

- The number of reaching nodes.
- The number of reaching nodes.
- The number of reaching nodes.
- The number of reaching nodes.
- The number of reaching nodes.
- The number of reaching nodes.

**Logical Functions**

-Logical functions are also available.

**Date-Time Functions**

- The date-time functions are also available.

**Spatial Functions**

- The spatial functions are also available.

**String Functions**

- The string functions are also available.

**Literal Functions**

- The literal functions are also available.

**Math Functions**

- The mathematical functions are also available.

**Collect Functions**

- The collect functions are also available.

**Match**

- The match functions are also available.

**Spatial Index**

- The spatial index functions are also available.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.

**Use Parameters**

- Use parameters instead of literals when possible. This will reduce the number of mapping and query expressions.

**Use Case**

- Use case functions to combine queries.