Cypher is the declarative query language for Neo4j, the world-leading graph database. Key principles and capabilities of Cypher are as follows:

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

You can try Cypher snippets live in the Neo4j Console at [console.neo4j.org](http://console.neo4j.org) or read the full Cypher documentation in the [Neo4j Manual](http://neo4j.com/whitepapers/cypher-manual/). 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. Newn props can be strings, numbers, booleans or arrays thereof. Cypher also supports maps and collections.

**Syntax**

Read Query Structure

```cypher
MATCH [[Optional MATCH] WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

Node patterns can contain labels and properties.

Any pattern can be used in MATCH.

Any WHERE clause is optional: 

1. MATCH (nlabels:Prop1) -n:Prop2
2. MATCH (nlabels:Prop1) -n:Prop2 WHERE n:Prop3
3. MATCH (nlabels:Prop1) -n:Prop2 -m:Prop4 WHERE n:Prop3

**Assign a path to n:**

```cypher
MATCH [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

A node pattern with a WHERE clause is optional. Optional pattern, n[...], is used for missing parts.

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

**Write-Only Query Structure**

```cypher
CREATE [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

Use a predicate to filter. Note that what is always part of a node:Prop, newProp, or oldProp clause. Putting it after a different clause in a query will alter what it does.

Write-Only Query Structure

```cypher
MATCH [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

A node pattern with a WHERE clause is optional. Optional pattern, n[...], is used for missing parts.

**Read-only Query Structure**

```cypher
MATCH [[Optional MATCH] WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

Node patterns can contain labels and properties.

Any pattern can be used in MATCH.

Any WHERE clause is optional: 

1. MATCH (nlabels:Prop1) -n:Prop2
2. MATCH (nlabels:Prop1) -n:Prop2 WHERE n:Prop3
3. MATCH (nlabels:Prop1) -n:Prop2 -m:Prop4 WHERE n:Prop3

**Assign a path to n:**

```cypher
MATCH [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

A node pattern with a WHERE clause is optional. Optional pattern, n[...], is used for missing parts.

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

**Write-Only Query Structure**

```cypher
CREATE [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

Use a predicate to filter. Note that what is always part of a node:Prop, newProp, or oldProp clause. Putting it after a different clause in a query will alter what it does.

Write-Only Query Structure

```cypher
MATCH [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

A node pattern with a WHERE clause is optional. Optional pattern, n[...], is used for missing parts.

**Read-only Query Structure**

```cypher
MATCH [[Optional MATCH] WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

Node patterns can contain labels and properties.

Any pattern can be used in MATCH.

Any WHERE clause is optional: 

1. MATCH (nlabels:Prop1) -n:Prop2
2. MATCH (nlabels:Prop1) -n:Prop2 WHERE n:Prop3
3. MATCH (nlabels:Prop1) -n:Prop2 -m:Prop4 WHERE n:Prop3

**Assign a path to n:**

```cypher
MATCH [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

A node pattern with a WHERE clause is optional. Optional pattern, n[...], is used for missing parts.

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

---

**Path**

Path from a node through relationships to any other node in the graph. The nodes in the path must be of different types.

```cypher
MATCH ()-[r:TYPE]-(n)
```

r:TYPE is a relationship of type TYPE. 

**Example:**

```cypher
MATCH ()-[r:KNOWS]-(n)
```

**HOP Function**

```cypher
HOP(n)
```

n is a node in the graph. HOP(n) returns the path to n, the nodes in the path are a collection.

**Relationships**

```cypher
r
```

r is a relationship. The relationship can be typed or untyped.

**Example:**

```cypher
MATCH ()-[r:KNOWS]-(n)
```

**Match Pattern**

```cypher
MATCH [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

A node pattern with a WHERE clause is optional. Optional pattern, n[...], is used for missing parts.

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

**Write-Only Query Structure**

```cypher
CREATE [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

Use a predicate to filter. Note that what is always part of a node:Prop, newProp, or oldProp clause. Putting it after a different clause in a query will alter what it does.

Write-Only Query Structure

```cypher
MATCH [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

A node pattern with a WHERE clause is optional. Optional pattern, n[...], is used for missing parts.

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

**Read-only Query Structure**

```cypher
MATCH [[Optional MATCH] WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

Node patterns can contain labels and properties.

Any pattern can be used in MATCH.

Any WHERE clause is optional: 

1. MATCH (nlabels:Prop1) -n:Prop2
2. MATCH (nlabels:Prop1) -n:Prop2 WHERE n:Prop3
3. MATCH (nlabels:Prop1) -n:Prop2 -m:Prop4 WHERE n:Prop3

**Assign a path to n:**

```cypher
MATCH [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

A node pattern with a WHERE clause is optional. Optional pattern, n[...], is used for missing parts.

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

**Path**

Path from a node through relationships to any other node in the graph. The nodes in the path must be of different types.

```cypher
MATCH ()-[r:TYPE]-(n)
```

r:TYPE is a relationship of type TYPE. 

**Example:**

```cypher
MATCH ()-[r:KNOWS]-(n)
```

**HOP Function**

```cypher
HOP(n)
```

n is a node in the graph. HOP(n) returns the path to n, the nodes in the path are a collection.

**Relationships**

```cypher
r
```

r is a relationship. The relationship can be typed or untyped.

**Example:**

```cypher
MATCH ()-[r:KNOWS]-(n)
```

**Match Pattern**

```cypher
MATCH [WHERE [node:property] [AND] [OR] SKIP [LIMIT] ]
```

A node pattern with a WHERE clause is optional. Optional pattern, n[...], is used for missing parts.
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, labeled elements and properties.
- 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 the full Cypher documentation in the Neo4j Manual for live graph models using Cypher check out the Cypher Refcard.

The Cypher Refcard is also available in PDF format.

Note: (values) denotes either literals, for ad hoc Cypher queries, or parameters, which is the best practice for applications.

String matching
- Use functions.
- Use comparison operators.
- Use logical operators.
- Use chain operators to combine predicates.

For more information, visit Query Tuning.