Neo4j Cypher Cheat Sheet

Just a bunch of cyphering I found online - all in one place for easy consu

Cypher Fundamentals

Store any kind of data using the following graph concepts:

- **Node**: Graph data records
- **Relationship**: Connect nodes (has direction and a type)
- **Property**: Stores data in key-value pair in nodes and relationships
- **Label**: Groups nodes and relationships (optional)

Browser editor

CLI

Examples: `:help` `:clear`

Match

Match node

```
MATCH (ee:Person)
WHERE ee.name = "Romeo"
RETURN ee;
```

- **MATCH** clause to specify a pattern of nodes and relationships
- **(ee:Person)** a single node pattern with label 'Person' which will as
- **WHERE** clause to constrain the results
- **ee.name = "Romeo"** compares name property to the value "Rome
- **RETURN** clause used to request particular results

Gets gets the id 5 and id 0 nodes and creates a `:KNOWS` relationship bet

Match nodes and relationships

```
MATCH (ee:Person)-[:KNOWS]-(friends)
WHERE ee.name = "Romeo"
RETURN ee, friends
```

- **MATCH** clause to describe the pattern from known Nodes to founc
- **(ee)** starts the pattern with a Person (qualified by WHERE)
- **-[:KNOWS]-** matches "KNOWS" relationships (in either direction)
- **(friends)** will be bound to Romeo's friends

Match labels

```
MATCH (n:Person)
RETURN n
```

or

```
MATCH (n)
WHERE n:Person
RETURN n
```

Match multiple labels

`:Car` **OR** `:Person` labels

```
MATCH (n)
WHERE n:Person OR n:Car
RETURN n
```

`:Car` **AND** `:Person` labels

```
MATCH (n)
WHERE n:Person:Car
RETURN n
```

Match same properties

```
MATCH (a:Person)
WHERE a.from = "Korea"
RETURN a
```

Returns every node (and their relationships) where there's a property `fr`

Match friends of friends with same hobbies

Johan is learning surfing, and wants to know any friend of his friends wh

```
MATCH (js:Person)-[:KNOWS]-()-[:KNOWS]-(surfer)
WHERE js.name = "Johan" AND surfer.hobby = "surf
```

```
RETURN DISTINCT surfer
```

- **()** empty parenthesis to ignore these nodes
- **DISTINCT** because more than one path will match the pattern
- **surfer** will contain Allison, a friend of a friend who surfs

Match by ID

Every node and relationship has an internal autonumeric ID, which can be
IN** operators:

Search node by ID

```
MATCH (n)
WHERE id(n) = 0
RETURN n
```

Search multiple nodes by ID

```
MATCH (n)
WHERE id(n) IN [1, 2, 3]
RETURN n
```

Search relationship by ID

```
MATCH ()-[n]-()
WHERE id(n) = 0
RETURN n
```

Create

Create node

```
CREATE (ee:Person { name: "Romeo", from: "Korea"
```

- **CREATE** clause to create data
- **()** parenthesis to indicate a node
- **ee:Person** a variable `ee` and label `Person` for the new node
- **{}** brackets to add properties (key-value pairs) to the node

Create nodes and relationships

```
MATCH (ee:Person) WHERE ee.name = "Romeo"
CREATE (js:Person { name: "Johan", from: "Korea"
(ir:Person { name: "Ian", from: "England", title
(rvb:Person { name: "Rik", from: "Belgium", pet:
(ally:Person { name: "Allison", from: "Californi
(ee)-[:KNOWS {since: 2001}]- (js),(ee)-[:KNOWS {
(js)-[:KNOWS]- (ir),(js)-[:KNOWS]- (rvb),
```

```
(ir)-[:KNOWS]- (js),(ir)-[:KNOWS]- (ally),
(rvb)-[:KNOWS]- (ally)
```

- **MATCH** clause to get "Romeo" in `ee` variable
- **CREATE** clause to create multiple nodes (comma separated) with t creates directed relationships `(a)-[:Label {key: value}]- (`

Create relationship between 2 unrelated nodes

```
MATCH (n), (m)
WHERE n.name = "Allison" AND m.name = "Romeo"
CREATE (n)-[:KNOWS]- (m)
```

Alternative with `MERGE`, which ensures that the relationship is created or

```
MATCH (n:User {name: "Allison"}), (m:User {name:
MERGE (n)-[:KNOWS]- (m)
```

Create node with multiple labels

```
CREATE (n:Actor:Director)
```

Update

Update node properties (add new or modify)

Add new `.owns` property or modify (if exists)

```
MATCH (n)
WHERE n.name = "Rik"
SET n.owns = "Audi"
```

Replace all node properties for the new ones

Danger: It will delete all previous properties and create `.plays` and `.a`

```
MATCH (n)
WHERE n.name = "Rik"
SET n = {plays: "Piano", age: 23}
```

Add new node properties without deleting old ones

Danger: If `.plays` or `.age` properties are already set, it will overwrite t

```
MATCH (n)
WHERE n.name = "Rik"
SET n += {plays: "Piano", age: 23}
```

Add new node property if property not already set

```
MATCH (n)
WHERE n.plays = "Guitar" AND NOT (EXISTS (n.like
```

```
SET n.likes = "Movies"
```

Rename a property in all nodes

```
MATCH (n)
WHERE NOT (EXISTS (n.instrument))
SET n.instrument = n.plays
REMOVE n.plays
```

Alternative

```
MATCH (n)
WHERE n.instrument is null
SET n.instrument = n.plays
REMOVE n.plays
```

Add label to existing node

Adds the `:Food` label to nodes id 7 and id 8

```
MATCH (n)
WHERE id(n) IN [7, 8]
SET n:Food
```

Creates the node if not exists and updates (or creates) a prope

```
MERGE (n:Person {name: "Rik"})
SET n.owns = "Audi"
```

Delete

Delete nodes

To **delete a node** (p.e. id 5), first we need to **delete its relationships**. T

```
MATCH (n)-[r]-()
WHERE id(n) = 5
DELETE r, n
```

To **delete multiple nodes** (must have their relationships previously delet

```
MATCH (n)
WHERE id(n) IN [1, 2, 3]
DELETE n
```

Deletes a property in a specific node

```
MATCH (n)
WHERE n:Person AND n.name = "Rik" AND n.plays is
REMOVE n.plays
```

Alternative

```
MATCH (n)
WHERE n:Person AND n.name = "Rik" AND EXISTS (n.
REMOVE n.plays
```

Delete a label from all nodes

Deletes the `:Person` label from **all** nodes

```
MATCH (n)
REMOVE n:Person
```

Delete a label from nodes with specific labels

Deletes the `:Person` label from nodes with `:Food` and `:Person` label

```
MATCH (n)
WHERE n:Food:Person
REMOVE n:Person
```

Delete multiple labels from nodes

Deletes the `:Food` and `:Person` labels from nodes which have **both** la

```
MATCH (n)
WHERE n:Food:Person
REMOVE n:Food:Person
```

Danger: Deletes the `:Food` and `:Person` labels from nodes which hav `:Food:Person` labels

```
MATCH (n)
REMOVE n:Food:Person
```

Delete entire database

```
MATCH (n)
OPTIONAL MATCH (n)-[r]-()
DELETE n, r
```

Other clauses

Show execution plan

Use `PROFILE` or `EXPLAIN` before the query

`PROFILE`: Shows the execution plan, query information and **db hits**. Exa 3.0, planner: COST, runtime: INTERPRETED. 84 total db hits in 32 ms.

`EXPLAIN`: Shows the execution plan and query information. Example: Cy planner: COST, runtime: INTERPRETED.

Count

Count all nodes

```
MATCH (n)
RETURN count(n)
```

Count all relationships

```
MATCH ()-- ()
RETURN count(*);
```

Limit

Returns up to 2 nodes (and their relationships) where there's a property

```
MATCH (a:Person)
WHERE a.from = "Korea"
RETURN a
LIMIT 2
```

Create unique property constraint

Make `.name` property unique on nodes with `:Person` label

```
CREATE CONSTRAINT ON (n:Person)
ASSERT n.name IS UNIQUE
```

Drop unique property constraint

Make `.name` property unique on nodes with `:Person` label

```
DROP CONSTRAINT ON (n:Person)
ASSERT n.name IS UNIQUE
```
```## Useful Cypher Queries for Neo4J


Find the unique labels that appear in the databa

```bash
match n
return distinct labels(n)
```

Find the unique relationships that appear in the database:

```
match n-[r]-()
return distinct type(r)
```

Combine the previous two queries to return the unique combinations rela
database:
```

```
match n-[r]-()
return distinct labels(n), type(r)
```

Find nodes that don't have any relationships:

```
start n=node(*)
match n-[r?]-()
where r is null
return n
```

Find all nodes that have a specific property:

```
start n=node(*)
match n
where has(n.someProperty)
return n
```

Find all nodes that have a specific relationship (regardless of the directio

```
start n=node(*)
match n-[:SOME_RELATIONSHIP]-()
return distinct n
```

Show the nodes and a count of the number of relationships that they hav

```
start n=node(*)
match n-[r]-()
return n, count(r) as rel_count
order by rel_count desc
```

Get a count of all nodes in your graph:

```
start n=node(*)
match n
return count(n)
```

To delete all nodes in a database (first you have to delete all relationship

```
start n=node(*)
match n-[r]-()
delete r

start n=node(*)
match n
delete n
```

A simple query to get nodes of a certain category that match a certain pr

```
match (n:Person) where n.name="Tim" return n
```

Useful Cypher Queries for Neo4J

Find the unique labels that appear in the database:

```
match n
return distinct labels(n)
```

Find the unique relationships that appear in the database:

```
match n-[r]-()
return distinct type(r)
```

Combine the previous two queries to return the unique combinations rela
database:

```
match n-[r]-()
return distinct labels(n), type(r)
```

Find nodes that don't have any relationships:

```
start n=node(*)
match n-[r?]-()
where r is null
return n
```

Find all nodes that have a specific property:

```
start n=node(*)
match n
where has(n.someProperty)
return n
```

Find all nodes that have a specific relationship (regardless of the directio

```
start n=node(*)
match n-[:SOME_RELATIONSHIP]-()
return distinct n
```

Show the nodes and a count of the number of relationships that they hav

```
start n=node(*)
match n-[r]-()
return n, count(r) as rel_count
order by rel_count desc
```

Get a count of all nodes in your graph:

```
start n=node(*)
match n
return count(n)
```

To delete all nodes in a database (first you have to delete all relationships

```
start n=node(*)
match n-[r]-()
delete r

start n=node(*)
match n
delete n
```

A simple query to get nodes of a certain category that match a certain pr

```
match (n:Person) where n.name="Tim" return n
```