Neo4j Cheatsheet

Basic Concepts

- **Node**: Represents an entity (e.g., person, city). It has properties and a label.
- **Relationship**: Represents a connection between nodes (e.g., `FRIEND_OF`, `LIKES`). Relationships have types and properties.
- **Property**: A key-value pair on nodes or relationships.
- **Label**: A tag on a node to categorize it.

1. Database Setup

- **Create a Database**:

```
:use system
CREATE DATABASE myDatabase
```

- **Switch to Database**:

```
:use myDatabase
```

2. Basic Cypher Syntax

Creating Data

- **Create a Node**:

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

- **Create a Relationship**:

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

Reading Data

- **Retrieve All Nodes**:

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

- **Filter Nodes by Label**:

```
MATCH (p:Person) RETURN p
```

- **Filter Nodes by Property**:

```
MATCH (p:Person {name: 'Alice'}) RETURN p
```

- **Retrieve Relationships**:

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

Updating Data

- **Update a Node's Property**:

```
MATCH (p:Person {name: 'Alice'})
SET p.age = 31
```

- **Update Multiple Properties**:

```
MATCH (p:Person {name: 'Alice'})
SET p += {age: 32, city: 'Pretoria'}
```

Deleting Data

- **Delete a Relationship**:

```
MATCH (a)-[r:FRIEND_OF]->(b)
DELETE r
```

- **Delete a Node** (Node must have no relationships):

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

- **Delete Node and Relationships**:

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

3. Advanced Querying

Pattern Matching

- **Find Friends of Friends**:

```
MATCH (a:Person)-[:FRIEND_OF]->(friend)-[:FRIEND_OF]->(fof)
WHERE a.name = 'Alice'
RETURN fof
```

- **Optional Match (for optional relationships)**:

```
MATCH (p:Person {name: 'Alice'})
OPTIONAL MATCH (p)-[:FRIEND_OF]->(friend)
RETURN p, friend
```

Filtering with WHERE

- **Conditions on Properties**:

```
MATCH (p:Person)
WHERE p.age > 25 AND p.city = 'Pretoria'
RETURN p
```

- **Check if Property Exists**:

```
MATCH (p:Person)
WHERE exists(p.city)
RETURN p
```

Aggregations

- **Count Nodes**:

```
MATCH (p:Person)
RETURN count(p)
```

- **Group By and Aggregate**:

```
MATCH (p:Person)
RETURN p.city, avg(p.age) AS average_age
```

4. Working with Paths

- **Find Shortest Path**:

```
MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'})
MATCH path = shortestPath((a)-[*]-(b))
```

```
RETURN path
```

- **All Paths of Length `n`**:

```
MATCH path = (a:Person)-[*2]-(b:Person)
RETURN path
```

5. Using Functions

- **String Functions**:

```
MATCH (p:Person)
RETURN p.name, toUpper(p.name) AS upper_name
```

- **Mathematical Functions**:

```
MATCH (p:Person)
RETURN p.name, round(p.age / 10.0) * 10 AS rounded_age
```

- **List Functions**:

```
MATCH (p:Person)
RETURN collect(p.name) AS all_names
```

6. Constraints and Indexes

- **Create a Unique Constraint**:

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

- **Create an Index**:

```
CREATE INDEX person_age_index FOR (p:Person) ON (p.age)
```

- **Drop Index/Constraint**:

```
DROP INDEX person_age_index
DROP CONSTRAINT unique_person_name
```

7. Transactions

- **Begin and Commit Transactions**:

```
BEGIN
CREATE (a:Person {name: 'Charlie'})
COMMIT
```

- **Rollback Transactions** (in case of errors):

```
BEGIN
CREATE (a:Person {name: 'Error Test'})
ROLLBACK
```

8. Common Use-Cases

- **Merge (Create if Not Exists, Update if Exists)**:

```
MERGE (p:Person {name: 'Alice'})
ON CREATE SET p.created = timestamp()
ON MATCH SET p.lastSeen = timestamp()
```

- **Finding Mutual Friends**:

```
MATCH (a:Person {name: 'Alice'})-[:FRIEND_OF]->(friend)-[:FRIEND_OF]->
(b:Person {name: 'Bob'})
RETURN friend
```

9. Data Import

- **Load CSV Data**:

```
LOAD CSV WITH HEADERS FROM 'file:///data.csv' AS row
CREATE (:Person {name: row.name, age: toInteger(row.age)})
```

10. Tips for Optimization

- **Use Indexes**: Make sure to index frequently queried properties.
- **Limit Results**: When exploring data, use `LIMIT` to avoid long waits.

```
MATCH (p:Person) RETURN p LIMIT 10
```

- **Avoid Using `*` in Matches**: Be specific in patterns for faster queries.
