# Cypher Query Language

Neo4j's Cypher language is purpose built for working with graph data.

- uses patterns to describe graph data
- familiar SQL-like clauses
- declarative, describing what to find, not how to find it

## CREATE

Create a node

```
CREATE (ee:Person { name: "Emil", from: "Sweden", klout: 99 })
```

- `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 to the node

## MATCH

Finding nodes

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

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

# CREATE more

Nodes and relationships

`CREATE` clauses can create many nodes and relationships at once.

```
MATCH (ee:Person) WHERE ee.name = "Emil"
CREATE (js:Person { name: "Johan", from: "Sweden", learn: "surfing" }),
(ir:Person { name: "Ian", from: "England", title: "author" }),
(rvb:Person { name: "Rik", from: "Belgium", pet: "Orval" }),
(ally:Person { name: "Allison", from: "California", hobby: "surfing" }),
(ee)-[:KNOWS {since: 2001}]->(js),(ee)-[:KNOWS {rating: 5}]->(ir),
(js)-[:KNOWS]->(ir),(js)-[:KNOWS]->(rvb),
(ir)-[:KNOWS]->(js),(ir)-[:KNOWS]->(ally),
(rvb)-[:KNOWS]->(ally)
```

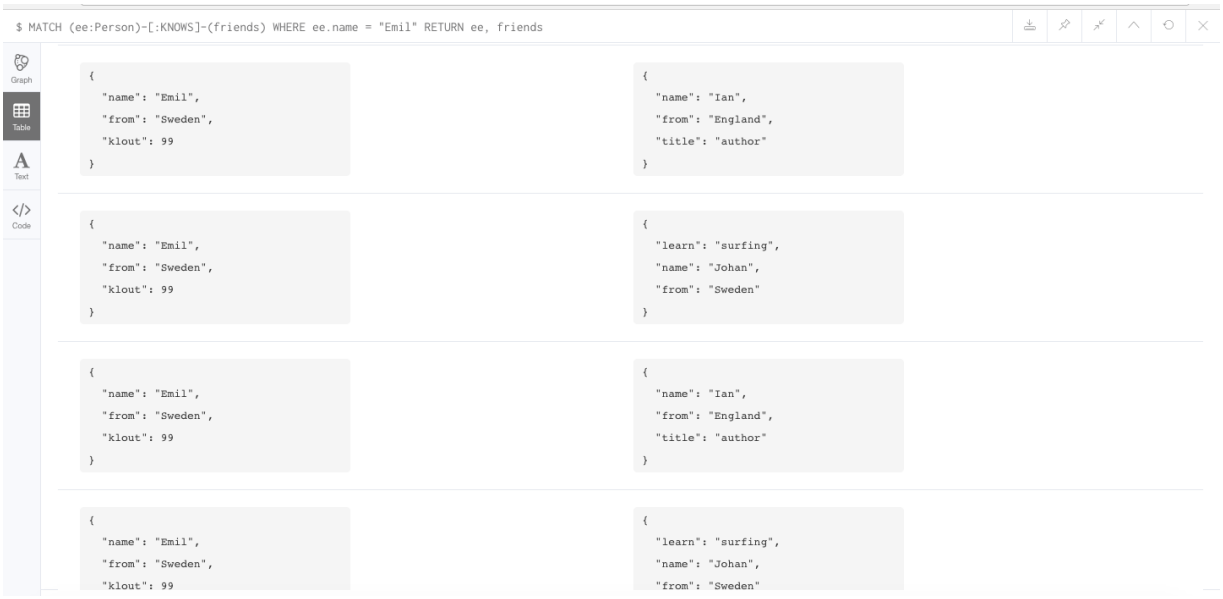
# Pattern matching

Describe what to find in the graph

For instance, a pattern can be used to find Emil's friends:

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

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




# Recommend

Using patterns

Pattern matching can be used to make recommendations. Johan is learning to surf, so he may want to find a new friend who already does:

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

# Analyze

Using the visual query plan

Understand how your query works by prepending `EXPLAIN` or `PROFILE`:

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