# graphdb gveGraph Databases

Graph databases are databases where stored data is represented as a graph. A graph contains nodes which are organized by relationships. Each node and relationship can have properties for storing unstructured information.

In comparison, a relational database management system (RDBMS) represent data as columns in tables and relationships are implemented via foreign keys. To navigate upon foreign-keys, join operations are required. A join operation is a cost-intensive operation because the RDBMS has to match two columns in order to properly link two columns.

A graph database is a NoSQL database which is specialized for highly connected information, like social media (social graph), online retailer (customers who bought this item also bought), etc. Navigating relationships does not require any special operations as relationships are first class member of a graph database. No joins are required.

Where relational algebra are the mathematical foundations of RDBMS, graph theory are the foundations of a graph database. This allows graph databases to fully integrate and optimized graph algorithms like the shortest path, spanning tree and Dijkstra algorithm.

## Neo4j

Neo4j is an implementation of a graph database used by big companies like Ebay, HP, Cisco, Walmart, etc. For our evaluation purposes we used the Neo4j Community edition, which runs local on a single machine. For use in production the Neo4j Enterprise edition offers enterprise-grade availability, management and scale-up & scale-out capabilities.

To connect to a Neo4j database, a wide range of technologies is offered. This includes following, but not complete list

* HTTP Post Requests,
* Java,
* .NET,
* JavaScript,
* Python.

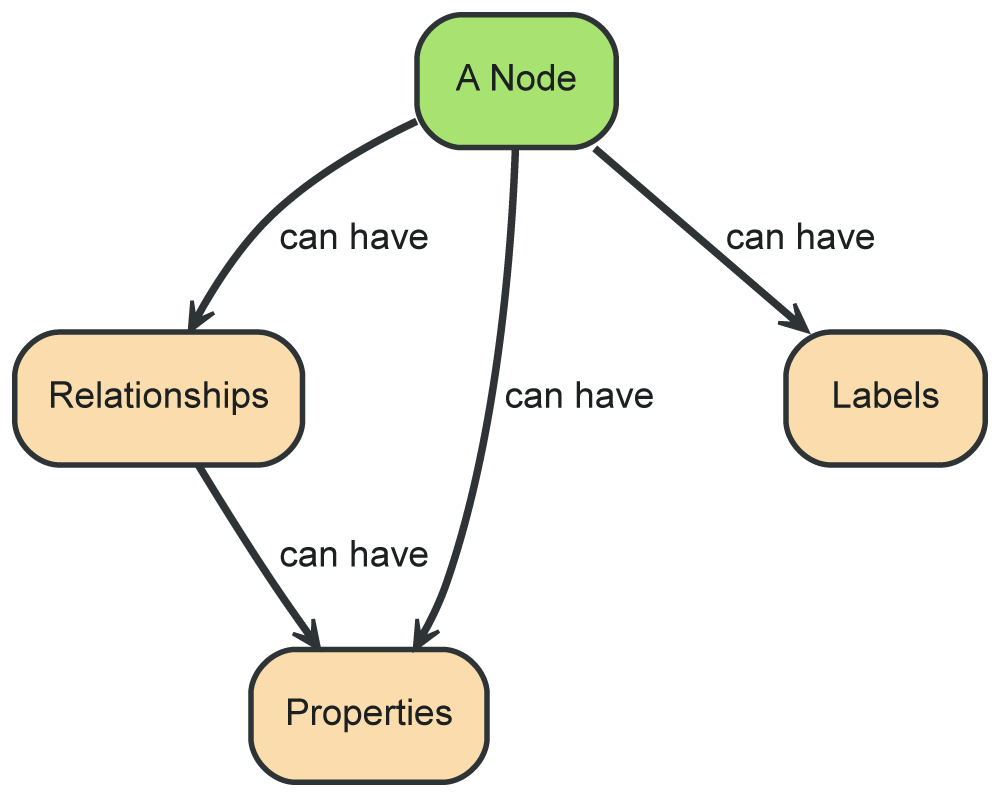
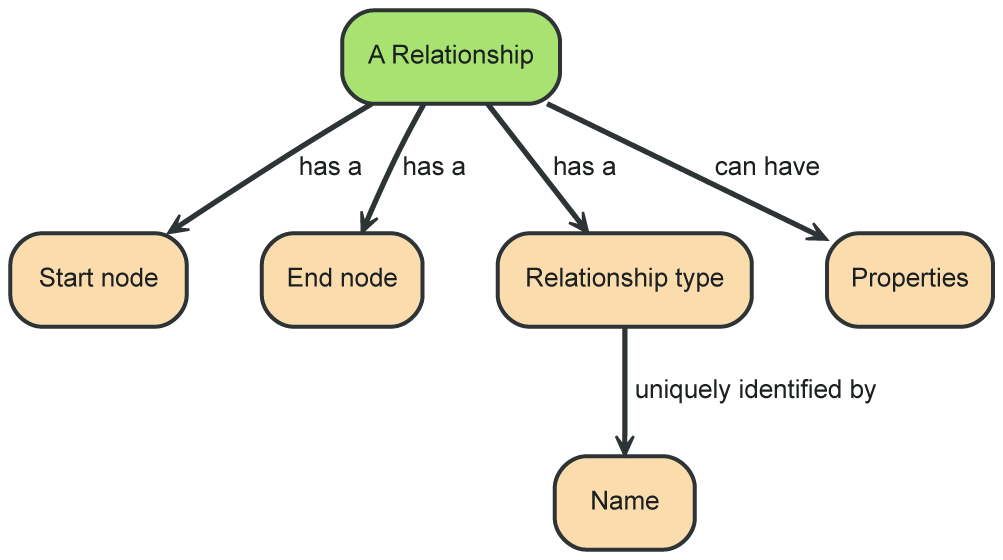
Even for less common languages like Clojure and Haskell, a Neo4j driver is offered.

Neo4j and its query language (see section Cypher) is very well documented online. The documentation contains general concepts of graph databases, the features and specifics of Neo4j, an easy to understand tutorial and a detailed reference about the Cypher query language.

### Graph Model

A Neo4j graph contains nodes and relationships. Both nodes and relationships can have properties where primitive values (Boolean, int, float, …), arrays and strings can be stored.

A node can have properties, relationships and can have a label. With labels, nodes can be grouped into certain sets, for instance types of objects. A relationship has a start node, an end node, a relationship-type and can also have properties. The relationship-type is used similar to a label for nodes. Neo4j also specifies a path, which has a start and an end node and can contain one or more relationships.



### Data Representation

In Neo4j, each entity physically stores a list of relationships. Therefore instead of costly joins, Neo4j can navigate along these relationships in constant time. In addition Neo4j can materialize relationships into database structures for a more efficient access.

|  |  |
| --- | --- |
| from relational model | to graph model |
| Storage of relationships in a RDBMS. | Storage of relationships in Neo4j. |

## Cypher

Neo4j provides an easy to use query language, called Cypher, which is used for creating, updating and deleting nodes, relationships and properties as well as for simple and complex queries. Similar do SQL, Cypher is a declarative language where de required information are described and not how to obtain those.

Cypher is designed to be a “humane query language” which is very natural and easy to use query language. Thereby the syntax of Cypher contains visual elements for relationships (arrows) to allow a visual interpretation of the query.

### CRUD Operations

For the creation of nodes and relationships in Cypher, the CREATE clause is used. A node is enclosed by parentheses and the properties within a nodes are enclosed by braces. Relationships are written as arrows (-->, <--) between matched nodes and the type and properties are enclosed by brackets. Even when Neo4j is storing both ends of a relationship physically, a logically navigability can be stated.

**CREATE** (p:Page {id:1, title:"Philosophy"})  
**CREATE** (l:Page {id:2, title:"Logic"})  
**CREATE** (l)-[:links\_to]->(p);

The most important clause of Cypher is the MATCH clause which performs a pattern matching. A matching node can have a name and a label and can specify properties that have to match. With the RETURN statement, all matching nodes are retrieved.

**MATCH** (p:Page {title:"Philosophy"})  
**RETURN** p;

Properties of matched nodes can be changed, created with the SET clause. When setting a non-existing property, the property is created. Properties can be deleted with the REMOVE clause.

**MATCH** (p:Page {title:"Philosophy"})  
**SET** p.test="Hello Graph"  
**REMOVE** p.length;

Nodes can be delete with the DELETE clause.

**MATCH** (p:Page {title:"Philosophy"})

**DELETE** p;

With the WHERE clause, the matching node can be further restricted.

**MATCH** (p:Page)  
**WHERE** p.length > 100  
**RETURN** p;

### Advanced Queries

Cypher supports advanced graph navigation in queries. Where in SQL, joins are necessary, Cypher provides an easy to use navigation using the arrow operator.

**MATCH** (p:Page {title:"Philosophy"})  
**MATCH** (p)<--(a)  
**RETURN** a;

Note in that the upper query that it is possible to only specify a single match and use the page-match and the link in a single MATCH statement. When combined in a single MATCH, the page will be matched multiple times, resulting in a less performant query.

Relationships can also be named and further restricted within brackets.

**MATCH** (p:Page {title:"Philosophy"})  
**MATCH** (p)<-[l:links\_to]-(a)  
**RETURN** a, l;

When multiple “hops” are possible for a match, the multiplicity can be easily stated.

**MATCH** (p:Page {title:"Philosophy"})  
**MATCH** (p)<-[l:links\_to\*1..3]-(a)  
**RETURN** a, l;

Even arbitrary long paths are easily stated. In SQL such statement require the definition of a recursive statement, which are often hard to write, read and maintain.

**MATCH** (p:Page {title:"Philosophy"})  
**MATCH** (p)<-[:links\_to\*]-(a)  
**RETURN** a;

Typical predicates like ALL, ANY, EXISTS exist, as well as aggregations like COUNT, statistic functions like SUM, AVG and MEAN.

**MATCH** (p:Page {title:"Philosophy"})  
**MATCH** (p)<-[:links\_to]-(a)  
**RETURN** COUNT(**DISTINCT** a);

Neo4j also supports graph algorithms for complex graph evaluation. Among others, Neo4j supports

* shortest path,
* Dijkstra and
* A\*.

For instance, the shortest path algorithm requires two matching nodes and a variable length relationship to obtain the shortest path between them.

**MATCH** path=shortestPath(  
 (p:Page {title:"Philosophy"})  
 <-[:first\_links\_to\*]-  
 (g:Page {title:"Graph"})  
) **RETURN** path;

The Dijkstra and A\* algorithm requires additional information, like the “costs-property”. In a RDBMS, these graph algorithms would be very hard or impossible to write.