Neo4j Cypher 快速参考

俞方桦 博士
(Joshua.yu@neo4j.com)

Neo4j Inc. APAC
2018年

1、Neo4j的标签属性图模型

Label Property Graph (LPG)

Neo4j 标签属性图模型

- Nodes – 节点。在其他图模型中称作“点”、“顶点”、“对象”。
- Relationships – 关系。在其他图模型中也称作“边”、“弧”、“线”。关系拥有类型。
- Properties – 属性，可以定义在节点和关系上。
- Labels – 标签，代表节点的类别。

标签属性图模型 – 示例

节点

- 代表图中的对象
- 节点可以有一个或多个标签，也可以没有标签

标签属性图模型 – 示例(续)

节点

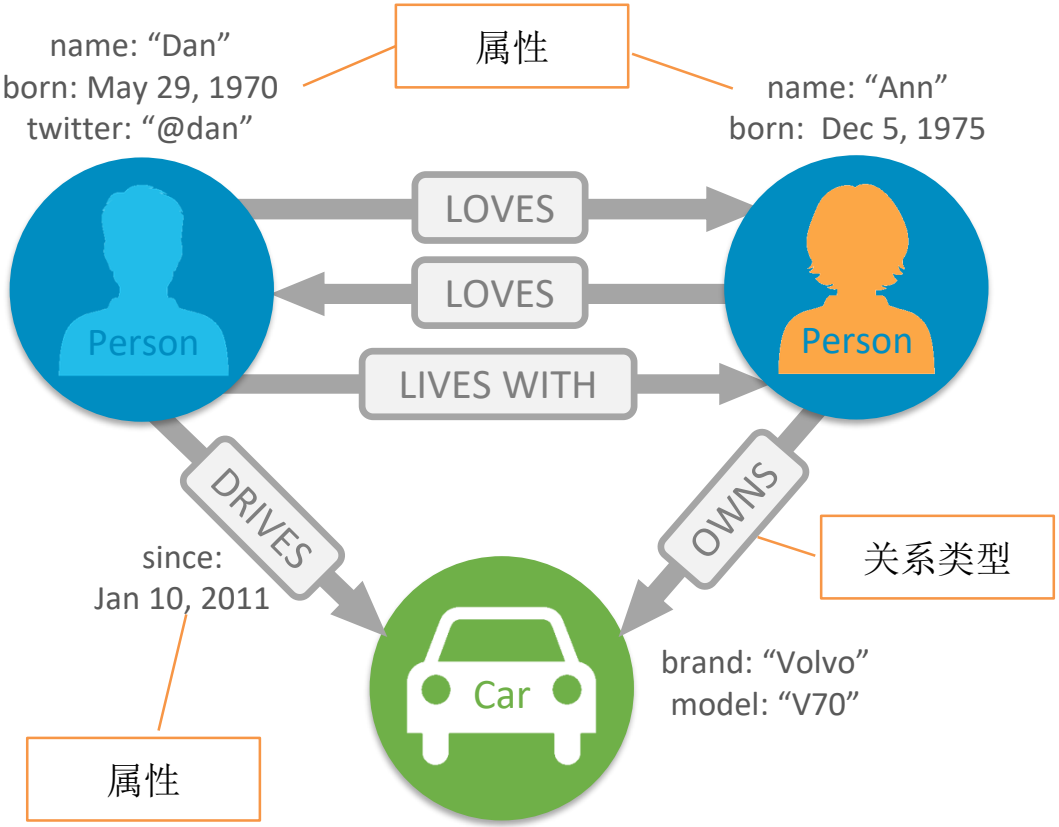
- 代表图中的对象
- 节点可以有一个或多个标签，也可以没有标签

关系

- 关系使用*类型*和*方向*将节点连结起来
- 在Neo4j中，关系必须是有向的
- 一个关系只连接两个节点
- 关系必须而且只能有一个类型

属性

- 键-值对，可以在节点和属性上定义。



小结

- **节点** – 实体或者复杂的数据类型
- **关系** – 连结实体、构造领域数据结构
- **属性** – 实体的属性、关系的量值、元数据
- **标签** – 按照角色、类型对节点分组

2、Neo4j的Cypher图查询语言

Cypher Graph Query Language

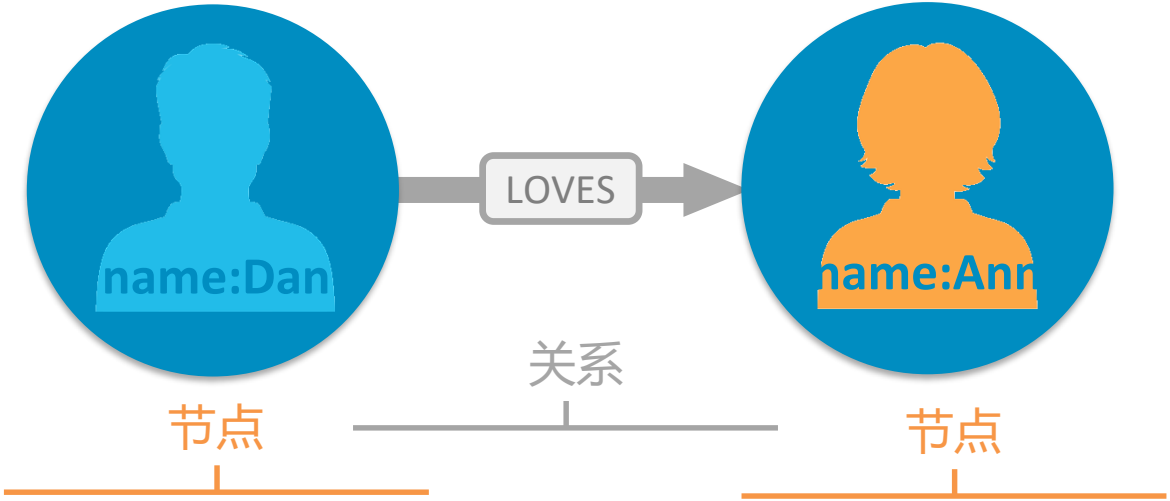
Cypher

专门为图数据库设计的、基于模式匹配的查询语言。

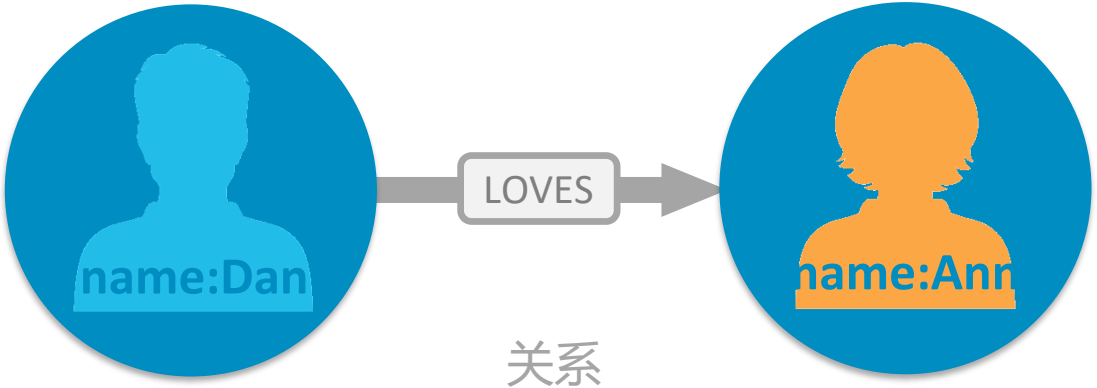
Cypher (续)

- Declarative / 声明型：定义要找的数据，Cypher会决定怎样是最优的查找方法。
- Expressive / 丰富的表达力：易于被开发人员和业务人员理解。
- Pattern Matching / 模式匹配：人类的思维更加容易识别和接受模式。

图数据模型中的“模式”



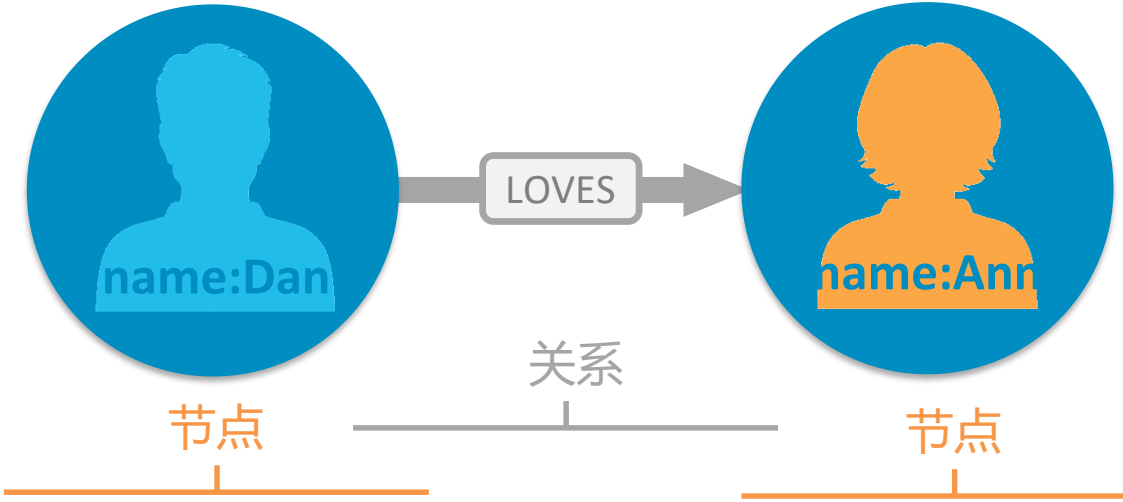
Cypher: 表达图数据模式




```
(:Person { name:"Dan"} ) -[:LOVES]-> (:Person { name:"Ann"} )
```

标签 属性 标签 属性

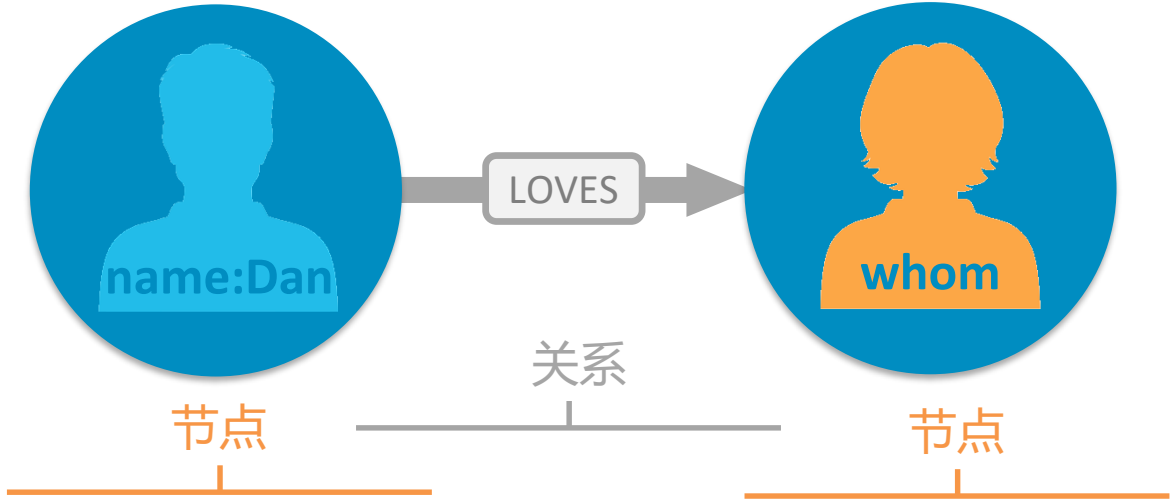
Cypher: CREATE/创建图数据模式




```
CREATE (:Person { name:"Dan"} ) -[:LOVES]-> (:Person { name:"Ann"} )
```

标签 属性 标签 属性

Cypher: MATCH / 匹配数据模式




```
MATCH (:Person { name:"Dan"} ) -[:LOVES]-> (whom:Person)
RETURN whom;
```

从Dan出发，沿着LOVES关系，返回所有能找到的Person节点。

3、Cypher的基本语法

Basic Syntax of Cypher

节点

节点必须包含在括号 () 内 。

```
(n:Label1:Label2)
```

- 标签名前必须有冒号‘:’
- 节点可以有多个标签
- 标签对节点进行分类，类似关系数据库中的表

```
(n)
```

- 节点可以没有或不指定标签。

```
(n:Label {prop: 'value'})
```

- 节点可以有属性

关系

关系两端各有一个短横线/减号，用方括号包含关系类型，关系类型名前面必须有冒号(:)。在其中一端用 > 或 < 代表关系的方向，也可以没有方向。下面都是合法的关系：

```
--, <-- , -->
-[:DIRECTED]->
```

关系(续)

--> 或 -[r:TYPE]->

- 关系以短划线/减号和方括号包含
- 与标签一样，关系类型前也必须有冒号‘:’

< > 指定关系的方向

- 关系在创建时必须指定方向
- 关系在查询时可以不指定方向(表示双向关系)

关系

```
-[:KNOWS {since: 2010}]->
```

- 关系也可以有属性！

模式

模式是由关系连接起来的节点构成的表达式，关系可以是有方向的，也可以是没有方向/双向的。

```
()-[]-()
()-[]->()
()<-[]-()
```

模式的例子

```
(n:Label {prop:'value'})-[:TYPE]->(m:Label)
```

- 最基本的模式：由一类关系连接的两个节点
- 使用变量保存匹配的结果

```
(p1:Person {name:'Alice'})-[:KNOWS]->(p2:Person {name:'Bob'})
```

如果存在从Alice到Bob的、类型为KNOWS的关系，那么上面的模式会将匹配的节点保存在p1和p2中。

Cypher查询的组成部分

```
MATCH (m:Movie)
RETURN m
```

MATCH 和 **RETURN** 是 Cypher 关键字

m 是变量，保存节点

:Movie 是标签

Cypher查询的组成部分(续)

```
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie)
RETURN p, r, m
```

MATCH 和 **RETURN** 是 Cypher 关键字
p 和 **m** 是变量，保存节点
r 是变量，保存关系
:Movie 是节点标签
:ACTED_IN 是关系类型

Cypher查询的组成部分(续)

```
MATCH path = (:Person)-[:ACTED_IN]->(:Movie)
RETURN path
```

MATCH 和 **RETURN** 是Cypher关键字

path 是变量，保存路径

:Movie 是节点标签

:ACTED_IN 是关系类型

图查询的结果 vs. 表状数据结果

```
MATCH (m:Movie)
RETURN m
```

返回匹配的节点列表，所有节点的标签都是Movie。

图查询的结果 vs. 表状数据结果(续)

```
MATCH (m:Movie)
RETURN m.title, m.released
```

属性以{variable}.{property_key}的方式访问。这个查询返回包含两列的表状数据：title/标题，和released/放映年份。

关于大小写

大小写敏感

节点标签

关系类型

属性名/键

大小写不敏感

Cypher 关键字

关于大小写 – 示例

大小写敏感

:Person

:ACTED_IN

name

大小写不敏感

MaTcH

return

命名规范

关键字 – 全部大写

MATCH, RETURN, …

标签名 – 首字母大写

n:Movie

关系名 – 全部大写

:ACTED_IN

属性名，变量名 – 小写

m.name