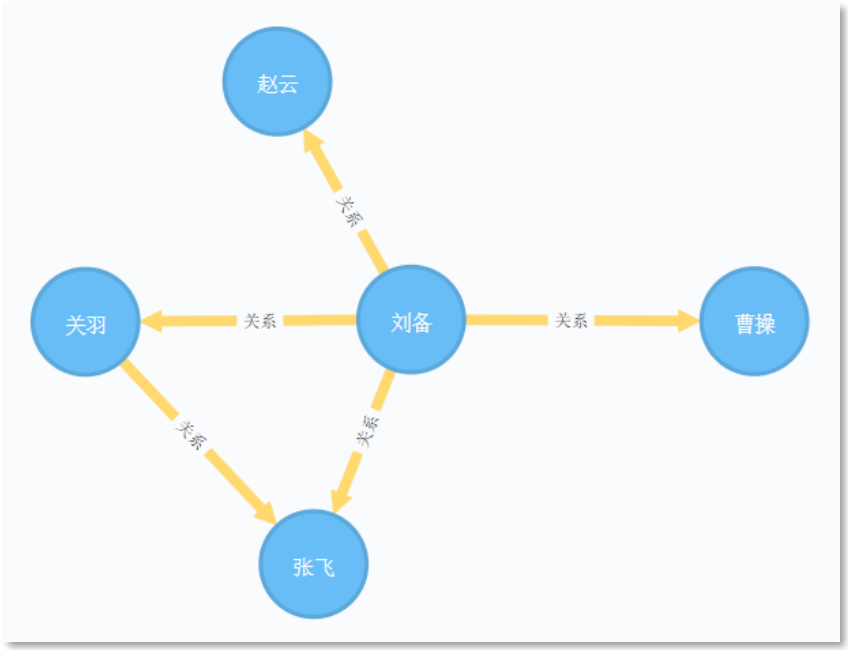
4、创建和删除节点、关系和属性

Create and Delete Nodes, Relationship and Property

创建一个简单的图数据库

下面，我们来创建一个简单的三国人物关系图：



创建一个简单的图数据库(续)

使用下面的Cypher语句：

```
CREATE (n1:`人物` {name: '刘备'}) -[:`关系`{relationship:'兄长'}]-> (n2:`人物` {name: '关羽'}),
       (n2) -[:`关系`{relationship:'兄长'}]-> (n3:`人物` {name: '张飞'}),
       (n1) -[:`关系`{relationship:'兄长'}]-> (n3),
       (n1) -[:`关系`{relationship:'主公'}]-> (n4:`人物` {name: '赵云'}),
       (n1) -[:`关系`{relationship:'对手'}]-> (n5:`人物` {name: '曹操'});
```

说明：

- 标签名和关系类型是非ASCII字符(例如中文)或者包含空格时，可以用反引号(`)包含起来；
- 使用变量n1、n2... 来保存和引用创建了的节点；
- 可以在创建节点和关系的同时创建属性；
- 关系必须是有向的；
- 这里使用通用的'关系'来代表人物之间的关系，仅作示例目的，并没有考虑实际建模的需要。

创建一个简单的图数据库(续)

在Cypher语句中：

- CREATE用来创建节点和属性
- 可以一次性创建多个节点和属性
- 可以使用变量保存创建的节点和属性
- 节点可以拥有一个或多个标签，也可以没有
- 关系必须指定关系类型

查询图数据

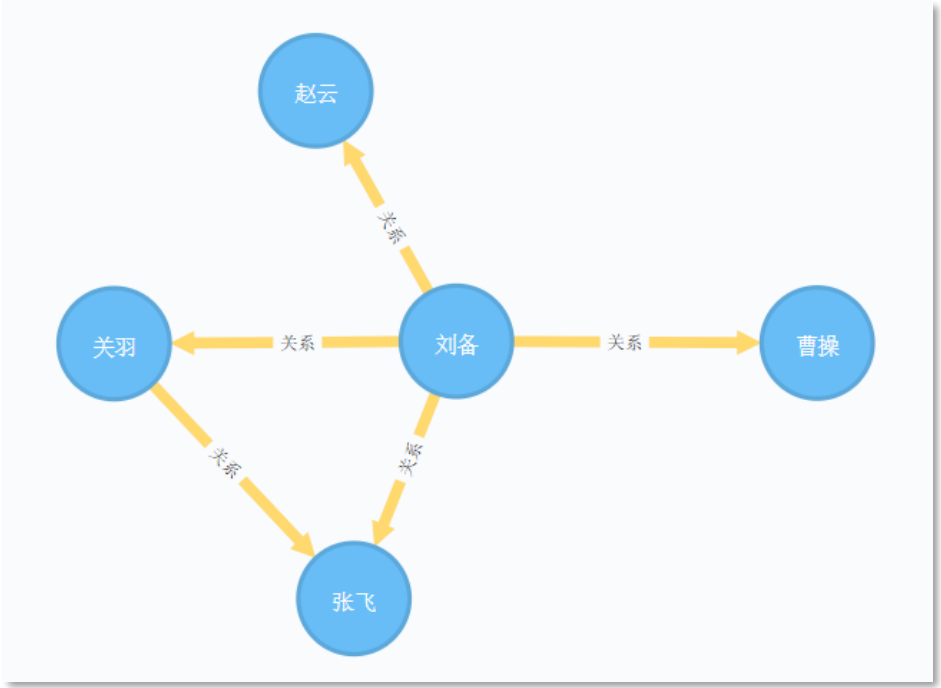
使用下面的Cypher语句：

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

或者：

```
MATCH path = () -- () RETURN path;
```

都会得到右边的结果。



查询图数据 (续)

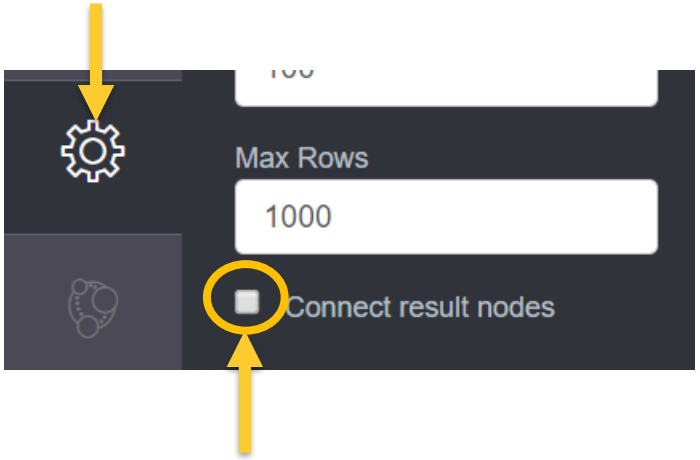
区别在于:

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

- 只返回所有节点
- Neo4j浏览器自动将节点之间的关系也从数据库中读取出来

```
MATCH path = () -- () RETURN path;
```

- 返回所有节点和关系



提示 可以取消这个选项让Neo4j浏览器不再自动读取结果节点中的关系。

更多查询的例子

```
// 寻找‘刘备’节点
MATCH (n:人物{name:'刘备'}) RETURN n;
```

等同于:

```
MATCH (n:人物) WHERE n.name = '刘备' RETURN n;
```

```
// 寻找刘备的兄弟
MATCH (n:人物{name:'刘备'}) -[:关系{relationship:'兄长'}]-> (n1) RETURN n1;
```

```
// 寻找刘备的小弟(兄弟的兄弟)
MATCH (n:人物{name:'刘备'}) -[:关系*2{relationship:'兄长'}]-> (n1) RETURN n1;
```

```
// 寻找刘备的所有社会关系。关系可以双向。
MATCH (n:人物{name:'刘备'}) -[:关系]- (n1) RETURN n1;
```

更多查询的例子(续)

如果说“兄弟的敌人是我的敌人”，试试看，能不能找到谁是张飞的敌人？

** 提示 ** 数据库中目前只有刘备拥有“对手”关系，因此从“张飞”出发，沿着“兄长”关系找到所有其他节点，然后找到这些节点的“对手”节点。另外，关系在查询时可以是反向，或者不指定方向的。例如：

```
MATCH (n:人物{name:‘张飞’}) <-[:关系{relationship:'兄长'}]- (n1) RETURN n1;
```

更改和显示节点标签

```
// 为‘刘备’增加2个标签
MATCH (n:人物{name:'刘备'}) SET n:皇帝:好人;
```

```
// 显示’刘备’节点的标签和属性
MATCH (n:人物{name:'刘备'}) RETURN labels(n) AS nodeLabel, keys(n) AS nodeProperties, n.name;
```

```
// 删除’刘备’的’好人’标签
MATCH (n:人物{name:'刘备'}) REMOVE n:好人;
```

```
// 显示节点的标签和属性，以及关系的类型和属性，并以表状结构返回结果
MATCH (n:人物{name:'刘备'}) -[r]- (n1)
RETURN type(r) AS relationshipType, keys(r) AS relationshipProperties, labels(n1) AS nodeLabel,
keys(n1) AS nodeProperties, n1.name;
```

更改属性

```
// 为‘刘备’增加性别属性
MATCH (n:人物{name:'刘备'}) SET n.genre = '男';
```

```
// 删除‘刘备’的性别属性
MATCH (n:人物{name:'刘备'}) SET n.genre = NULL;
```

节点的属性在第一次被赋值时创建。拥有相同标签的节点可以不一定拥有完全一样的属性。属性值为空(NULL)时数据库会删除该属性。

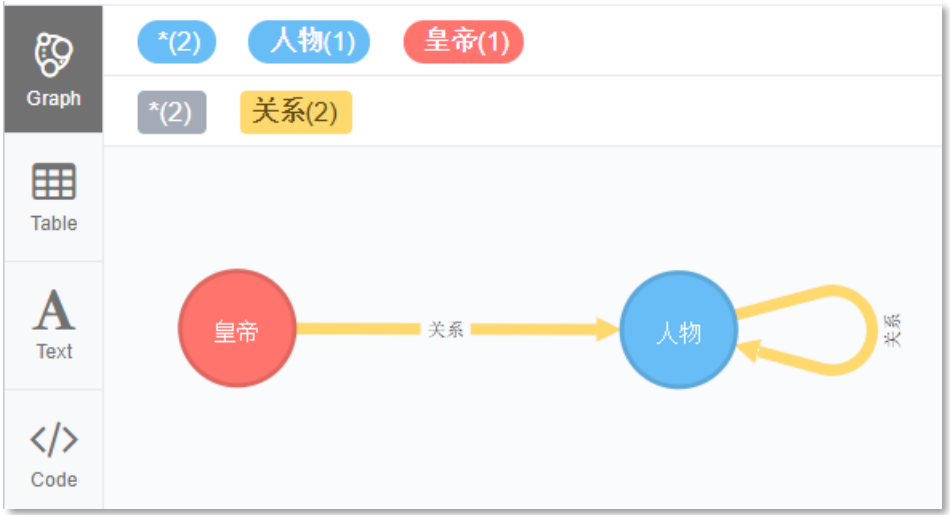
对关系的属性更改也是如此。

显示图数据库中的元模型

```
// 显示数据库的元模型(meta graph model)
CALL db.schema;
```

** 提示 **

- Neo4j的数据库模式在创建节点和关系时自动被创建;
- 元模型展示标签之间的关系类型;
- 使用apoc.meta.graph过程可以现实图的元模式，同时还包括每种标签和关系类型的统计数据。关于apoc，搜索“Neo4j+APOC”以获得更多信息。



删除节点和关系

```
// 如果节点没有关系
MATCH (n:人物{name:'刘备'}) DELETE n;
```

```
// 如果节点有关系，使用DETACH DELETE会先删除连接到该节点的所有关系，然后再删除节点
MATCH (n:人物{name:'刘备’}) DETACH DELETE n;
```

```
// 删除所有节点。如果节点有关系，也删除关系。
MATCH (n) DETACH DELETE n;
```

```
// 仅删除关系，保留节点
MATCH (n:人物{name:'刘备'}) -[r]- () DELETE r;
```

** 提示 ** 删除全库最简单的方法是删除Neo4j安装目录下的data/databases/graph.db文件夹。当然，这需要先停止数据库服务。

关于节点的id

```
// 根据数据库中的节点id寻找节点
MATCH (n) WHERE id(n) = 0 RETURN n;
```

- 节点在创建时会被赋予一个数据库的内部id。该id类似于关系数据库表中的“行ID”(rowid)，不可修改，也不可指定。
- Id永远是正整数;
- 根据内部id寻找节点可以使用id(node)函数;
- 由于Neo4j使用固定长度块存储节点，使用id访问节点是最快速的方法。
- 节点也可以有自己的id属性，例如从业务需求而来的数据标识。该属性可以被赋值、修改，并以node.id的形式访问。

总结

- Neo4j是’Schema-lite’(轻型模式)的图数据库，因此不需要事先定义标签、或者关系类型；
- 标签在第一次创建节点时被定义；
- 关系类型在第一次创建关系时被定义；
- Neo4j在查询时**不会**做模式校验，也就是说如果标签、关系类型或者属性不存在，查询不会返回“对象不存在”的错误，而只是无法查询到任何结果。例如下面的查询：

  MATCH (n:人物1{name:'刘备'}) -[:关系1]-> (n1) RETURN n1;
- Neo4j数据库不存储值为NULL的属性

总结 (续)

- 在对数据库中节点和关系规模不清楚时，谨慎使用下面的查询：

```
MATCH (n:人物) RETURN n;  // 相当于SELECT * FROM 人物
MATCH (n) RETURN n;       // 相当于SELECT * FROM *
MATCH path = () -[*]- () RETURN *;  // 相当于查询所有节点，以及所有路径组合
```

以避免造成查询结果过多而导致服务停止响应。

- 可以在RETURN 后面使用LIMIT限制返回的数据量：

```
MATCH (n:人物) RETURN n LIMIT 10;
```