图数据库Neo4j

为什么要使用图数据库?

1. 关系型数据库的弊端

   在rdbms中，通过外键约束来实现量表或者多个表之间某些记录相互引用的关系。外键约束是关系型数据库中实现表之间相互引用的必不可少的策略。通过外键在主表中寻找匹配的主键记录来进行搜索，匹配计算操作，这种操作是“计算密集型”的，也是“内存密集型”的，并且操作次数将是表中记录的指数级别，所以它需要消耗大量的系统资源，如果使用多对多关系的话，更是要维护一张中间表，进一步增加了连接的成本。

2. 图数据库的优势

   在图数据库中，关系是最重要的元素，通过关系我们能将节点相互关联起来构建与我们的问题领域密切相关的复杂模型。

   图数据库模型中的每个节点都直接包含一个关系列表，关系列表中存放次节点与其他节点的关系记录。这些关系记录按类型和方向组织起来，并且可以保存附加属性，无论何时运行类似关系数据库的连接**(join)** 操作时，图数据库都将使用此列表来直接访问连接的节点，无需进行搜索和匹配计算。这种能力能够提供比关系行数据库高几个数据量级的性能，特别是对于复杂连接查询，neo4j能够实现毫秒级别的响应。

3. 图数据库完全ACID事务，包括`wal`预写式日志。

免索引邻接

Neo4j使用免索引邻接来保证关系查询的速度，免索引邻接即： 数据库中的每个节点都会维护与他相临节点的引用。听起来跟链表一样。

这样比使用全局索引的代价要小的多，这意味着查询时间和图的整体规模无关，只与他附近的节点的数量成正比。

常用语句

给节点的某个属性添加唯一性约束

```
CREATE CONSTRAINT ON (n :Gakkiyomi) ASSERT n.name IS UNIQUE;
```

根据两个已有节点创建一条关系并返回

```
MATCH (a),(b) WHERE a.name="fangc" AND b.name="ssg229" CREATE (a) -[x:Layer3
{vlan: 131}]-> (b) return x;
```

节点添加新属性

```
MATCH (n:Gakkiyomi) WHERE n.id = "a22" SET n.sex = "男" RETURN n
```

删除指定关系

```
MATCH (p1:Gakkiyomi)-[r:Friend {id:"349f8fa3-65b7-4c53-b047-1d6c3aa5ec8c"}]-
(p2:Firewall)
DELETE r
```

修改节点label

```
match(n:oldlabel) set n:newlabel remove n:oldlabel
```

修改关系label

```
MATCH p=(n:User)-[r:subord]->(e:department) create(n)-[r2:subord_new]->(e)
 set r2 = r with r delete r
```

根据默认生成id删除节点

```
MATCH (r),(b) WHERE id(r) = 10 AND id(b) = 9 Delete r,b
```

查询一条关系(返回 节点-关系-节点)

```
MATCH p=()-[r:Friend {id:"2ec0ddde-0eb1-4f6b-a9e4-094cfbdfc694"}]->() RETURN p
```

查询一条关系(只返回关系)

```
MATCH p=()-[r:Friend ]->() RETURN r
```

查询一条关系(返回关系和节点label)

```
MATCH p=(a)-[r:Friend]->(b) with p as ps, labels(a) as x,labels(b) as y return
ps,x,y
```

查询label名

```
MATCH (r:Firewall) RETURN distinct labels(r)
```

查询两点之间的最短路径 (3 为在路径深度为3以内中查找)

```
match(n{name:"哈士奇"}),(m{name:"fangc"})with n,m match p=shortestpath((n)-[*]->(m)) return p;
```

```
match(n{name:"哈士奇"}),(m{name:"ssg229"})with n,m match p=shortestpath((n)-[*..3]-(m)) return p;
```

`shortestpath`查询一条

`allshortestpath`查询所有

查询两点之间的所有路径

```
MATCH p=(a)-[*]->(b)
RETURN p
```

查询数组里的属性 [1,2,4,5]

```
match (n) where 5 in n.ip return n
```

修改节点属性

```
MATCH (a:Ta{names:"afaf"})
SET a.names="a"
return a
```

修改节点属性名称

```
match(n) set n.propertyNew=n.propertyOld remove n.propertyOld
```

查询多label多条件

```
match (n) where any(label in labels(n) WHERE label in ['HDSStorage','BrocadePort']) and '192.168.1.106' in n.ip or n.domain = '28' return n
```

Cypher语句规则和具备的能力:

`Cypher通过模式匹配图数据库中的节点和关系，来提取信息或者修改数据。`

`Cypher语句中允许使用变量，用来表示命名、绑定元素和参数。`

`Cypher语句可以对节点、关系、标签和属性进行创建、更新和删除操作。`

`Cypher语句可以管理索引和约束。`

运算符

常规运算	DISTINCT, ., []
算数运算	+, -, *, /, %, ^
比较运算	=, <>, <, >, <=, >=, IS NULL, IS NOT NULL
逻辑运算	AND, OR, XOR, NOT
字符串操作	+
List操作	+, IN, [x], [x .. y]
正则操作	=~
字符串匹配	STARTS WITH, ENDS WITH, CONTAINS

变长路径检索

变长路径的表示方式是：[*N...M]，N和M表示路径长度的最小值和最大值。
(a)-[*2]->(b)：表示路径长度为2，起始节点是a，终止节点是b；
(a)-[*3...5]->(b)：表示路径长度的最小值是3，最大值是5，起始节点是a，终止节点是b；
(a)-[*...5]->(b)：表示路径长度的最大值是5，起始节点是a，终止节点是b；
(a)-[*3...]->(b)：表示路径长度的最小值是3，起始节点是a，终止节点是b；
(a)-[*]->(b)：表示不限制路径长度，起始节点是a，终止节点是b；

查询所有节点的属性

```
match (n) return distinct keys(n)
```

neo4j 数据导入

	create 语句	**load csv 语句**	**Batch Inseter**	**Batch Import**	**neo4j-import**
适用场景	1~1w nodes	1w~10w nodes	千万以上 nodes	千万以上 nodes	千万以上 nodes
速度	很慢 (1000 nodes/s)	一般 (5000 nodes/s)	非常快(数万 nodes/s)	非常快(数万nodes/s)	非常快(数万nodes/s)
优点	使用方便，可实时插入。	使用方便，可以加载本地	远程CSV；可实时插入	基于Batch Inserter，可以直接运行编译好的jar包；可以在已存在的数据库中导入数据	官方出品，比Batch Import占用更少的资源
缺点	速度慢	需要将数据转换成csv	需要转成CSV；只能在JAVA中使用；且插入时必须停止neo4j	需要转成CSV；必须停止neo4j	需要转成CSV；必须停止neo4j；只能生成新的数据库，而不能在已存在的数据库中插入数据