http://www.cnblogs.com/hwaggLee/p/5959716.html

Neo4j中不存在表的概念，只有两类：**节点(Node)和关联(Relation)**，可以简单理解为图里面的点和边。  
在数据查询中，节点一般用**小括号()，关联用中括号[]**。  
当然也隐含路径的概念，是用节点和关联表示的，如：(a)-[r]->(b)，表示一条从节点a经关联r到节点b的路径。

**备份Neo4j的数据:**

**1**)停掉数据库.

**2**)备份D:\Neo4J\neo4j-enterprise-**1.9**.**1**\data目录下graph.db目录中的所有内容.

**3**)在服务器上拷贝graph.db目录中的内容到新的服务器的相同目录中,启动即可.

**Cypher的基本操作**

**1）创建节点**

create (a) 创建空节点

create (a:Person) 创建标签（可以理解为类）为Person的节点

create (a:Person {name:‘Kaine‘,age:28}) 创建标签为Person，属性name值为Kaine，属性age值为28的节点

**2）创建关联**

match (a),(b)

where a.name=‘Kaine‘ and b.name=‘Sharon‘

create (a)-[r]->(b) 创建a节点和b节点的路径，此时变量r即代表关联，它也可以有标签

**3）查询关键字**

match：用来匹配一定模式，可以是简单的节点、关联，也可以是复杂的路径

where：用来限定条件，一般是限定match中的出现变量的属性

return：返回结果

start：开始节点，一般用于有索引的节点或者关联

match ... where ... return ...

如果match有多个对象，用逗号隔开；

如果where有多个条件，用and连接；

如果return有多个变量，用逗号隔开

**4）查询举例讲解**

[复制代码](javascript:void(0);)

match (n) return n   
 查询所有节点及关联

match (a)-[r]->(b) where a.name=‘Kaine‘ return a,b   
 查询属性name的值是Kaine的节点，及其所有关联节点

match (a)-[\*1..3]->(b) where a.name=‘Kaine‘ return a,b   
 查询属性name值是Kaine的节点，及其所有距离为1到3的关联节点，

match (a)-[\*2]->(b) where a.name=‘Kaine‘ and not (a)-[\*1]->(b) return a,b   
 查询属性name的值是Kaine的节点，及其所有距离为2并且去除距离为1的节点。  
 （在计算好友的好友时会用到，即如果a、b、c三个人都认识，如果仅计算跟a距离为2的人的时候会把b、c也算上，因为a->b->c，或者a->c->b都是通路）

注：关联的中括号内数字的含义

n 距离为n

..n 最大距离为n

n.. 最小距离为n

m..n 距离在m到n之间

[复制代码](javascript:void(0);)

a.创建

CREATE (id:label {key:value})  
id: 为节点添加一个唯一ID，不设置则系统自动设置一个，不设置时是 CREATE (:label...  
label: 标签，生命节点类型  
{}: 属性定义，key/value格式

b.关系

-[role:label {roles: ["Neo"]}]->

-- 表示一个无指向的关系

--> 表示一个有指向的关系

[] 能够添加ID，属性，类型等信息

 另看：http://blog.csdn.net/wangweislk/article/details/47661863

[复制代码](javascript:void(0);)

按id查询(这里的id是系统自动创建的)：

start n=node(**20**) return m;

查询所有节点：

start n=node(\*) return n;

查询属性，关系：

start n=node(**9**) return n,n.name,n.ID,n.level; //查看指定节点，返回需要的属性

start n=node(\*) match (n)-[r:SubClassOf]->m return m,n,n.name,n.ID,r; //查找指定关系

按关系查询多个节点：

start a = node(**14**) match b-[r]<->a return r,b;

start a = node(**0**) match c-[:KNOWS]->b-[:KNOWS]->a return a,b,c; //查找两层KNOWS关系的节点

start a = node(**21**) match b-[\*]->a return a,b; //查找所有与a节点有关系的节点

使用Where条件进行查询：（不用建立Index也可以使用）

start n=node(\*) where n.name="Activity" return n;

并且可以使用特定符号：

start n=node(\*) where n.ID?="A\*" return n;

start n=node(\*) where HAS(n.type) return n,n.name,n.ID,n.type; //如果存在属性type，并且以A开头，就输出节点。

配置文件自动建立索引：

修改conf目录下的neo4j.properties文件内容如下，重启Neo4J，对重启后新建的Node生效。

# Enable auto-indexing for nodes, default is false

node\_auto\_indexing=true

# The node property keys to be auto-indexed, if enabled

node\_keys\_indexable=name,ID

# Enable auto-indexing for relationships, default is false

relationship\_auto\_indexing=true

# The relationship property keys to be auto-indexed, if enabled

relationship\_keys\_indexable=KNOWS,SubClassOf

建立索引后可以用node\_auto\_index按属性值查询：

start n=node:node\_auto\_index(name="C") return n,n.name;

修改属性值：

start a = node(\*) where a.name="a" set a.name="A" return a,a.name ;

start n=node(**0**) set n.name="Root",n.ID="**001**" ; //给默认的根节点添加name,ID属性，便于查询。

删除所有节点和关系：

MATCH (n)  
OPTIONAL MATCH (n)-[r]-()  
DELETE n,r

**2,查询**

match (p: Person) return p; 查询Person类型的所有数据

match (p: Person {name:"sun"}) return p; 查询名字等于sun的人

match( p1: Person {name:"sun"} )-[rel:friend]->(p2) return [p2.name](http://p2.name/) , p2.age 查询sun的朋友的名字和年龄

match (old) ... create (new) create (old)-[rel:dr]->(new) return new 对已经存在的节点和新建的节点建立关系