**[Neo4j之Cypher学习总结](https://www.cnblogs.com/zhaohuanhuan/p/10637640.html" \o "发布于 2019-04-01 17:21)**

**Cypher 语句**

Cypher是图形数据库Neo4j的声明式查询语言。  
**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 |

**语句**

**1.创建节点语句**

例子1:创建节点并给节点分配一个属性

CREATE (n {name:"张三"})

例子2:批量创建节点并且分配属性

UNWIND [{name:"李四"},{name:"张三"}] AS mynodes

CREATE (n) SET n = mynodes

例3:创建节点时分配属性和标签

CREATE ( n:Person{name:"李四"} )

例4:创建一个关系，给两个节点建立关系指定关系类型、方向和绑定一个变量。

CREATE ({name:"李四"})-[r:have]->({bookname:"设计模式"})

例5:创建两个节点同时创建一个关系，给两个节点建立关系指定关系类型、方向和属性。

CREATE ({name:"李四"})-[:LOVES{since:"2017"}]->({name:"小花"})

例6:给已经存在的两个节点创建一个关系。

MATCH (a:Person),(b:Person)

WHERE a.name = '张三' AND b.name = 李四'

CREATE (a)-[r:FRIEND]->(b)

RETURN r

**2.匹配语句**

Cypher的MATCH语句和SQL的SELECT语句类似,都是查询数据的表达式.而Cypher语句中的WHERE和SQL语句中的WHERE语句由这相同的作用,其作用是用来过滤匹配结的,在Cypher语句中WHERE是在MATCH、OPTIONAL MATCH、WITH或START语句中使用的过滤限定语句

*匹配语句结构*  
**[MATCH WHERE]  
[OPTIONAL MATCH WHERE]  
[WITH [ORDER BY] [SKIP] [LIMIT]]  
RETURN [ORDER BY] [SKIP] [LIMIT]**

例1:根据属性匹配节点信息。

MATCH (n{name:"张三"}) RETURN n

例2:根据标签和属性匹配节点信息。

MATCH (n:Person{name:"李四"}}) RETURN n

例3:任意匹配，返回任意节点信息。  
Cypher语句允许任意匹配，任意匹配可以匹配任何与cypher表达式相符合的所有节点.  
1.匹配任意节点

//该语句会匹配任意节点，并返回

MATCH (n) RETURN n

2.任意关系

//该语句是匹配出有任意关系的两个节点

MATCH p = (n)-[r]->(m) RETURN p

3.可选匹配

//匹配结果集中如果有丢的部分，则会用null来补充

OPTIONAL MATCH (n)-[r]->(m) RETURN m

例4:匹配节点时指定标签、属性和关系类型

MATCH (n:Person)-[:FRIEND]->(m:Person) WHERE n.name = '张三'

例5:根据标签过滤

//从图数据库中,过滤出Person标签

MATCH (n) WHERE n:Person RETURN n

例6:根据节点属性过滤

MATCH (n) WHERE n.name = "李思思" RETURN n

例7:根据关系属性过滤

MATCH (n)-[k:KNOWS]->(f) WHERE k.since < 2000 RETURN f

例8:字符串匹配  
字符串首尾可以使用STARTS WITH 和 ENDS WITH,去匹配定位字符串，字符串匹配还可以使用AONTAINS来匹配字符串中包含的自字符串。  
1.匹配字符串的开头  
匹配图中的所有节点，返回节点中包含name属性，且属性值以"张"开头的所有节点

MATCH (n)

WHERE n.name STARTS WITH '张'

RETURN n

2.匹配字符串的末尾  
匹配图中的所有节点，返回节点中包含name属性，且属性值以"三"开头的所有节点

MATCH (n)

WHERE n.name ENDS WITH '三'

RETURN n

3.匹配字符串的包含匹配  
匹配图中的所有节点，返回节点中包含name属性，且属性值中任意位置包含"小"字符串的节点

MATCH (n)

WHERE n.name CONTAINS '三'

RETURN n

4.字符串排除匹配  
在字符串匹配的时候可以使用NOT关键字来排除匹配到的结果，得到相反的结果

//匹配图中的所有节点，返回节点中包含name属性，且不是姓张的所有节点

MATCH (n)

WHERE NOT n.name STARTS WITH '张'

RETURN n

例9:字符串正则表达式  
Cypher支持使用正则表达式进行过滤。 正则表达式语法从Java正则表达式继承过来的。这包括支持改变字符串匹配的标志，包括不区分大小写(?i)，多行(?m)和逗点(?s),在这里正则表达式使用的解析符号是=~。

1.模糊匹配，类似sql语句中的like

//类似like '%小%'

MATCH (n)

WHERE n.name =~ '.\*小.\*'

RETURN n

2.不去分大小写正则匹配

//不区分大小写匹配，名字中以"ANDR"开头的节点

MATCH (n)

WHERE n.name =~ '(?i)ANDR.\*'

RETURN n

例10:WHERE句子中使用路径匹配  
1.使用逻辑运算符号和路径作为过滤条件

//在图中匹配名称为"李四"或"王五"与"张三"有任何关系的节点，并返回符合匹配条件的节点信息

MATCH (n { name: '张三' }),(m)

WHERE m.name IN ['李四', '王五'] AND (n)<--(m)

RETURN m

2.关系匹配使用NOT逻辑运算符号

//匹配和张三没有关系的人

MATCH (persons),(zhangsan { name: '张三' })

WHERE NOT (persons)-->(zhangsan)

RETURN persons

3.关系类型匹配，并使用正则表达式

//在图中匹配张三节点和所有节点的关系类型以F开头的所有关系

MATCH (n)-[r]->()

WHERE n.name='张三' AND type(r)=~ 'F.\*'

RETURN r

例11:使用"或"逻辑匹配关系

//返回任意关系为KNOWS和LOVES的所有节点关系

MATCH p=(n)-[:KNOWS|:LOVES]->(m) RETURN p

例12:关系深度匹配  
1.匹配从n到m，任意关系，深度1到5的节点

MATCH p=(n)-[\*1..5]->(m) RETURN p

2.匹配从n到m，任意关系、任意深度的节点

MATCH p=(n)-[\*]->(m) RETURN p

例13:匹配分页返回

MATCH (n) WHERE n.productName='苹果' RETURN n SKIP 10 LIMIT 10

例14:匹配去重返回

MATCH (n) WHERE n.productName='苹果' RETURN DISTINCT n

例15:匹配分组返回值

MATCH (n) WHERE n.productName='苹果' RETURN n ORDER BY n.price DESC

例16:别名返回

MATCH (n) WHERE n.productName=~'.\*苹果.\*' RETURN n.productName AS name

例17:WITH语句使用使用示例  
WITH语句是将第一部分匹配的输出作为下一部分匹配的输入。  
1.查找有十个以上朋友的张姓男子

MATCH (user)-[:FRIEND]-(friend)

WHERE user.name =~ '张.\*'

WITH user, count(friend) AS friends

WHERE friends > 10

RETURN user

2.WITH语句中使用ORDER BY、SKIP 和 LIMIT 语句示例。

MATCH (user)-[:FRIEND]-(friend)

WITH user, count(friend) AS friends

ORDER BY friends DESC

SKIP 1

LIMIT 3

RETURN user

例17:UNION语句使用使用示例  
1.使用UNION结果去重，求并集

MATCH (a)-[:KNOWS]->(b)

RETURN b.name

UNION

MATCH (a)-[:LOVES]->(b)

RETURN b.name

2.使用UNION ALL结果不去重, 求和集

MATCH (a)-[:KNOWS]->(b)

RETURN b.name

UNION ALL

MATCH (a)-[:LOVES]->(b)

RETURN b.name

例17:start语句匹配节点信息  
1.匹配所有节点

START n=node(\*) RETURN n

2.根据节点的具体id匹配

//在node函数中的值，是节点在图中的id

START n=node(1) RETURN n

**3.节点更新和删除**

例1:更新和添加一个属性

//1.创建一个节点，只有一个属性name='张三'

CREATE ({name:'张三'})

//2.给创建的"张三"节点，添加一个属性age=20，修改name属性值为"李四"

MATCH (n {name:'张三'})

SET n.name='李四'

SET n.age=20

RETURN n

例2: 修改节点信息,覆盖节点属性

//该语句会直接删除掉name属性，新增age属性

MATCH (n {name:'张三'})

SET n={age:20}

例3:修改节点信息,新增节点属性

//该语句不会删除掉name属性，而是在节点中新增age属性

MATCH (n {name:'张三'}) SET n += {age:20} RETURN n

例4:删除节点属性

MATCH (n{name:'张三'}) remove n.age RETURN n

例5:为节点添加新标签

//添加一个标签

MATCH (n{name:'张三'}) SET n:Person RETURN n

//添加多个标签

MATCH (n{name:'张三'}) SET n:Person:Student RETURN n

例6:删除节点标签

MATCH (n{name:'张三'}) REMOVE n:Person RETURN n

例7:为已经存在的两个节点添加关系

MATCH (a:Person {name: "张三"}),

(b:Person {name: "李四"})

MERGE (a)-[r:FRIENDS]->(b)

例8:匹配一个节点，更新属性如果属性不存在则创建该属性

MERGE (n:Person {name: "张三"})

ON CREATE SET n.created = timestamp()

ON MATCH SET

n.counter = coalesce(n.counter, 0) + 1,

n.accessTime = timestamp()

例9:匹配一个节点为他创建一个新的关系

//如果没有"王五"节点则创建一个新的

MATCH (a:Person {name: "张三"})

MERGE

(a)-[r:KNOWS]->(b:Person {name: "王五"})

例10:删除一个节和关系

MATCH (a)-[r:KNOWS]->(b) DELETE r,b

例11:级联删除

//删除"王五"节点之后，将与王五节点建立关系也删除掉

MATCH (n{name: "王五"}) DETACH DELETE n

**4.索引和约束**

例1:创建索引

//为"Person"标签的name属性创建索引

CREATE INDEX ON :Person(name)

例2:删除索引

//删除"Person"标签的name属性的索引

DROP INDEX ON :Person(name)

例3:匹配条件中使用索引

MATCH (n:Person)

WHERE n.name = 'Andres'

RETURN n

例4:创建节点属性唯一约束  
创建一个唯一约束保证图数据库中同一个标签内的节点的属性值不重复。  
注意：不可以为已经创建索引的属性创建唯一约束

CREATE CONSTRAINT ON (n:Person) ASSERT n.name IS UNIQUE

例5:创建节点属性唯一约束

DROP CONSTRAINT ON (n:Person) ASSERT n.name IS UNIQUE