报告6

目录

[Redis 2](#_Toc9771624)

[实验步骤 2](#_Toc9771625)

[存储结构 2](#_Toc9771626)

[查询1 2](#_Toc9771627)

[查询2 3](#_Toc9771628)

[查询3 4](#_Toc9771629)

[存储结构2 6](#_Toc9771630)

[查询3优化 6](#_Toc9771631)

[查询4 7](#_Toc9771632)

[心得 8](#_Toc9771633)

[MongoDB 9](#_Toc9771634)

[实验步骤 9](#_Toc9771635)

[存储结构 9](#_Toc9771636)

[查询1 9](#_Toc9771637)

[查询2 10](#_Toc9771638)

[查询3 11](#_Toc9771639)

[查询4 12](#_Toc9771640)

[心得 14](#_Toc9771641)

[Neo4j 14](#_Toc9771642)

[实验步骤 14](#_Toc9771643)

[存储结构 14](#_Toc9771644)

[LOAD CSV 15](#_Toc9771645)

[查询1 15](#_Toc9771646)

[查询2 16](#_Toc9771647)

[查询3 17](#_Toc9771648)

[查询4 18](#_Toc9771649)

[心得 20](#_Toc9771650)

[三者比较 20](#_Toc9771651)

[Appendix 21](#_Toc9771652)

[Reddis 21](#_Toc9771653)

[load\_triples 21](#_Toc9771654)

[查询1 22](#_Toc9771655)

[查询2 23](#_Toc9771656)

[查询3 23](#_Toc9771657)

[存储结构2 25](#_Toc9771658)

[查询4 26](#_Toc9771659)

[MongoDB 27](#_Toc9771660)

[存储结构 27](#_Toc9771661)

[查询1 28](#_Toc9771662)

[查询2 29](#_Toc9771663)

[查询3 30](#_Toc9771664)

[查询4 32](#_Toc9771665)

[Neo4j 33](#_Toc9771666)

[存储结构 33](#_Toc9771667)

[LOAD CSV 35](#_Toc9771668)

[查询1 35](#_Toc9771669)

[查询2 36](#_Toc9771670)

[查询3 37](#_Toc9771671)

[查询4 38](#_Toc9771672)

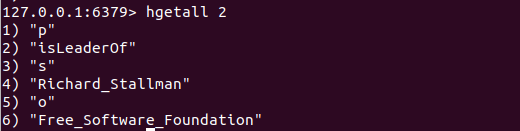
# Redis

## 实验步骤

### 存储结构

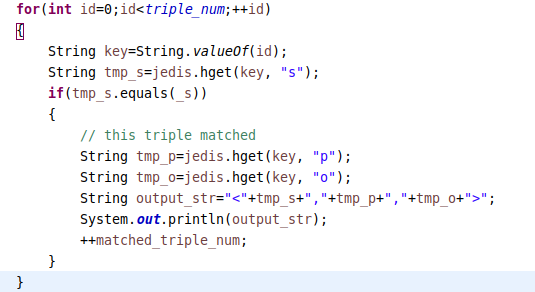
用哈希表存储，每个三元组对应一个对象key为行号，field为s, p, o和对应值的键值对，共100万个哈希表。

如图所示为第二个三元组对应的哈希表

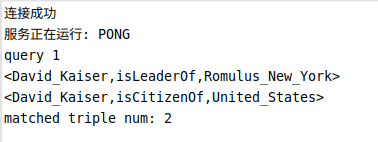


### 查询1

遍历每个哈希表，查看s域的值是否为给定的主语，如果是则找到一个匹配，打印这个三元组。详见附录。

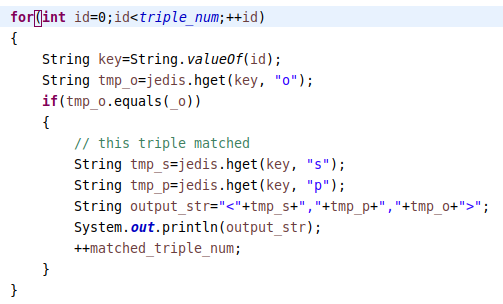


查询<David\_Kaiser,p,o>

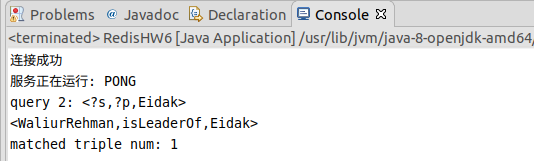


### 查询2

和查询1类似，遍历每个哈希表即每个三元组，如果宾语等于给定宾语，则打印该三元组。

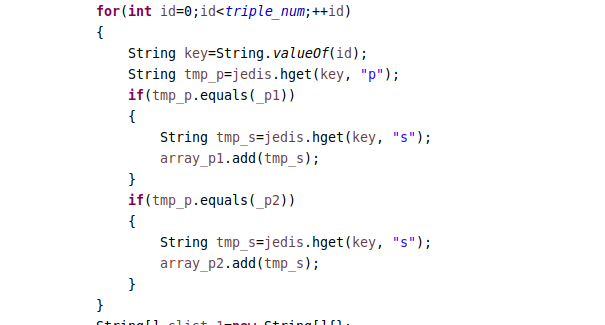


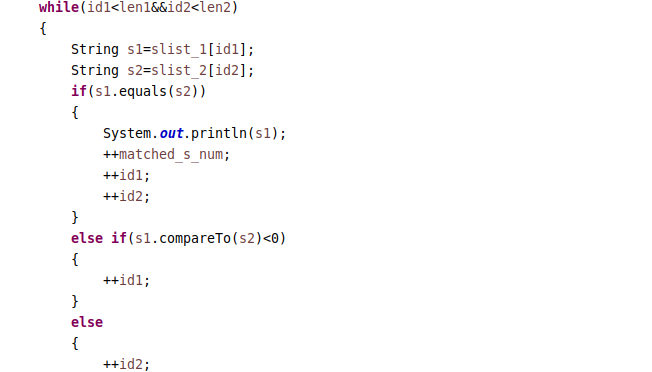
query\_2("Eidak");



### 查询3

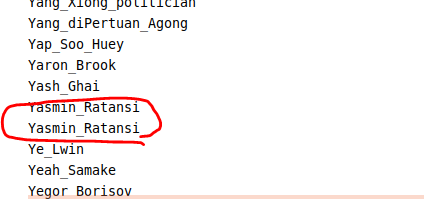
遍历每个三元组，如果谓词为p1则将主语加到主语序列1中，如果谓词为p2则将主语加到主语序列2中。对主语序列1和主语序列2排序，对两个序列从左向右遍历求交集即为结果。代码见附录。





query\_3("isLeaderOf","isCitizenOf")





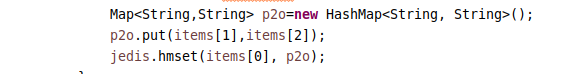
由于一个主语可能有多个相同的谓词，结果可能有重复的主语，所以排序前需要先去掉主语序列中的重复元素。

### 存储结构2

查询3只需要查找同时又p1,p2的主语，与主语对应的p1,p2个数无关，与宾语无关。故可以针对查询3的特性改变存储结构来实现优化。

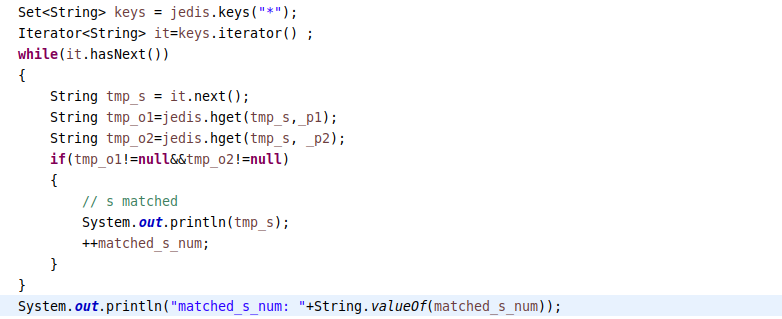
将subject作为哈希表的key，field为<predicate,object>的键值对，如图所示：



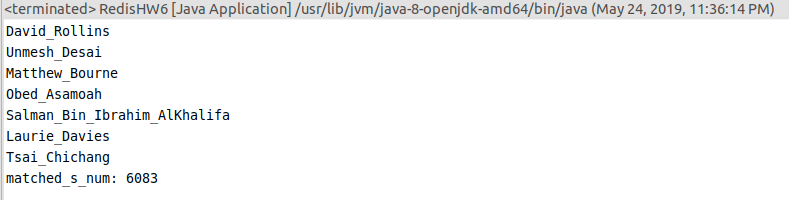


### 查询3优化

此时，查询3遍历每个哈希表，如果该哈希表存在p1和p2域，则这个主语符合要求



query\_3\_optimize("isLeaderOf","isCitizenOf");



如果存在多个三元组有相同的主语和谓语，则只剩下最后一个三元组，对查询结果没有影响，而且节省了存储空间。

另外，由于以subject为哈希表的key而key不重复，所以结果没有重复。结果6083就是不重复的主语，而之前的算法结果是6340包括了重复的主语

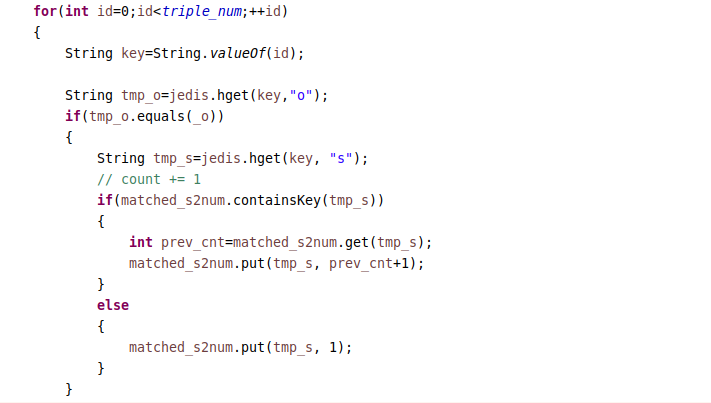
### 查询4

给定一个oi, 给出拥有这样oi最多的S

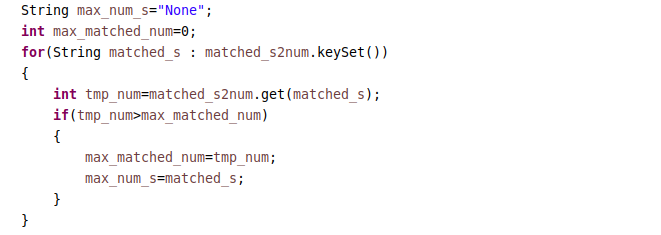
因为存储结构2的s,p重复项会覆盖，而查询4需要计数，故使用存储结构1。

用map<String,Integer>存储oi匹配到的主语分别有几个oi。遍历每个三元组的哈希表，如果宾语为给定的oi，则给该主语的计数加一。

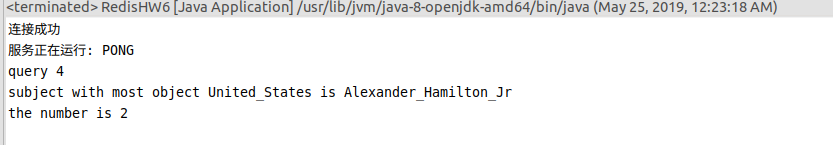




查找匹配数最多的主语



query\_4("United\_States")



代码见附录

## 心得

如果每个对象是一个key对应三个field即key为行号，field为s, p, o和对应值的键值对，则需要重复存储100万个“s","p","o",以及100万个行号，这些信息都是冗余或不必要的。同时s,p,o没有意义，给查询带来不便。

为了节省空间开销，尝试存储结构2.令每个三元组对应的对象只有一个key和一个field即主语作为key，谓语和宾语组成的键值对为一个field，但是存在重复的主语和重复的宾语的三元组，这些三元组都对应同一个哈希表，而该哈希表中域的键值重复，存在覆盖的情况，所以这不是适用于所有查询的结构。所以存储结构2适合存在性判断，不适合计数。

由于数据集较大，需要尽可能减少冗余的信息以及节约存储空间。在查询时，需要尽可能只遍历1次以节省时间。

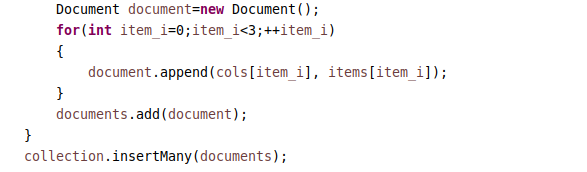
Redis的哈希表是无序的，可能有较高的访存开销

# MongoDB

## 实验步骤

### 存储结构

在test数据库下创建yago集合，文件的每个三元组以s，p，o为key组成一个文档，把1000000个文档加入到yago集合中。



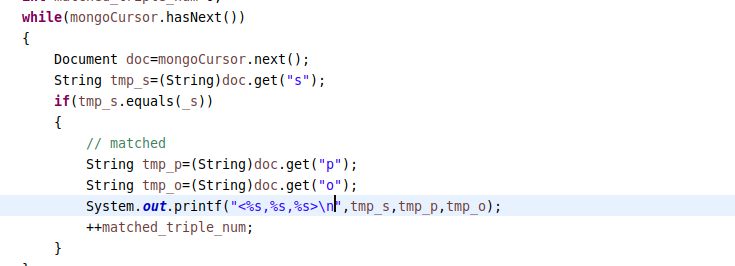
如图所示



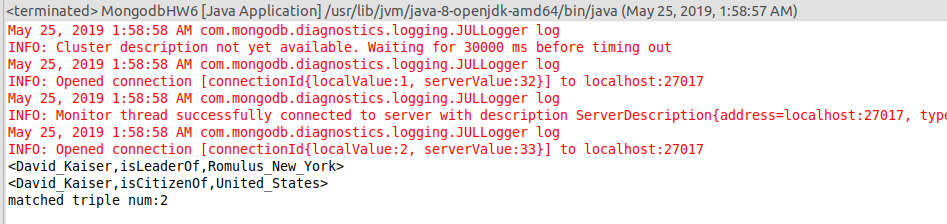
### 查询1

给定一个si，给出它所有的P和O，<si, P, O>

遍历每个document，如果主语为si则符合条件



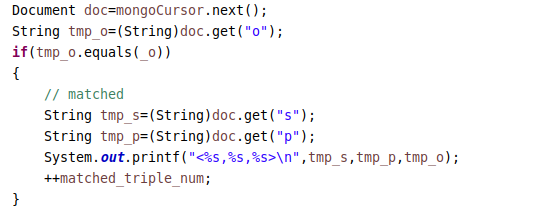
query\_1("David\_Kaiser")



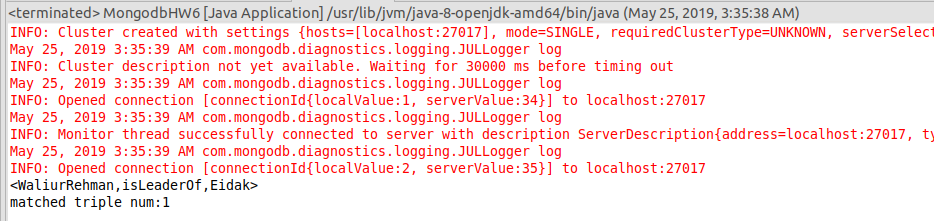
### 查询2

给定一个oi, 给出它所有的S和P，<S, P,oi>

遍历所有document，如果宾语为oi则符合要求



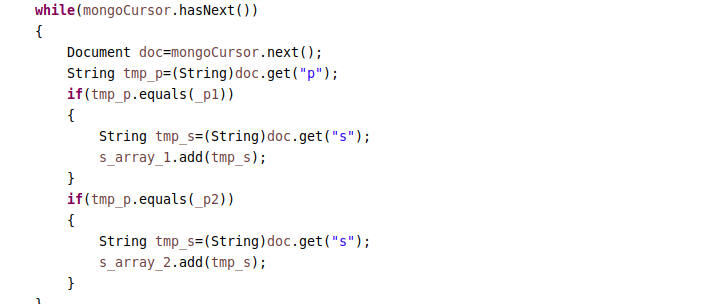
query\_2("Eidak");



### 查询3

给定两个p1,p2, 给出同时拥有它们的S，<S, p1, \*>, <S, p2, \*>

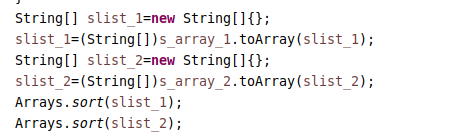
遍历每个document,如果谓语为p1，则将主语放进数组1中，如果谓语为p2，则将主语放进数组2中。



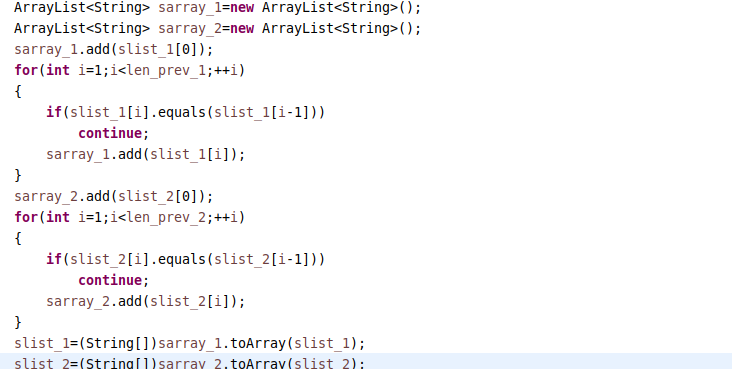
数组1和数组2的交集即为结果。求交集前，先对两个数组排序并去除重复元素。

在添加元素时，使用ArrayList，在排序时使用String[].

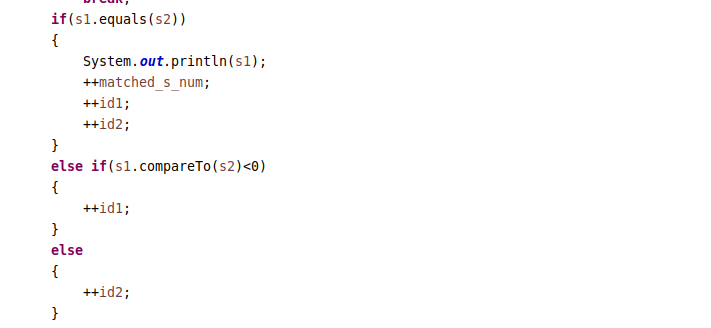
排序



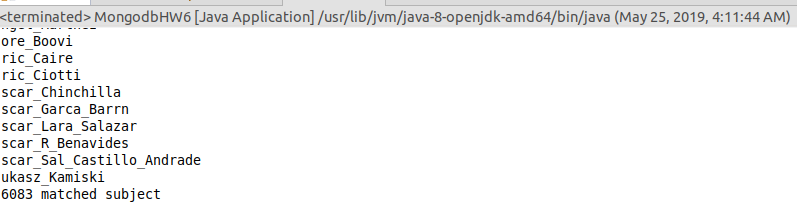
去除重复元素



取交集



query\_3("isLeaderOf","isCitizenOf")结果

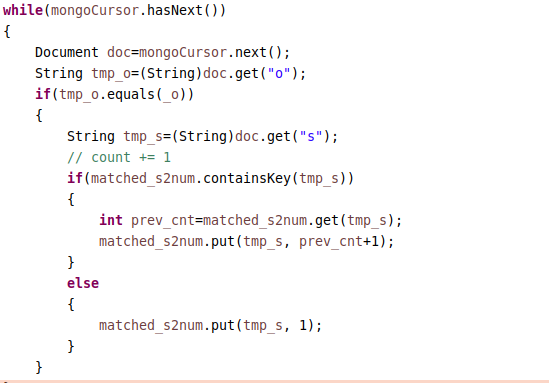


### 查询4

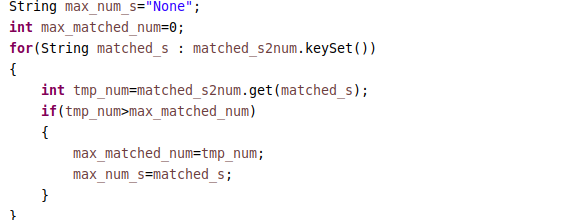
给定一个oi, 给出拥有这样oi最多的S

用map<String,Integer>存储oi匹配到的主语分别有几个oi。遍历每个三元组的哈希表，如果宾语为给定的oi，则给该主语的计数加一。

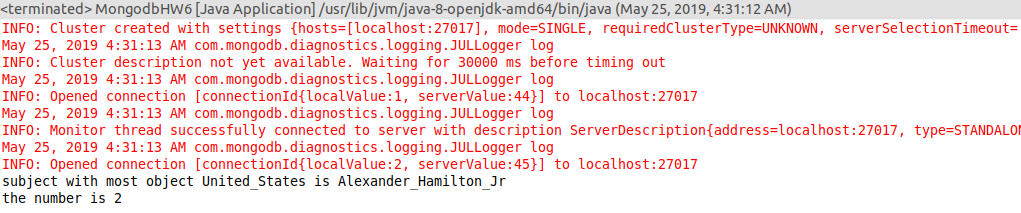




寻找匹配次数最多的s



query\_4("United\_States")



## 心得

Mongodb 以binary JSON 存储，最小单位为文档，只能逐文档搜索，不能只取一部分，搜索时可能有较高的时间开销。

但是由于只能顺序检索文档，避免了随机访存，所以导入数据和查询时间都比redis更短。

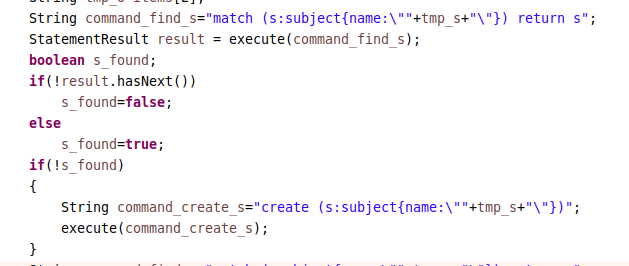
因为mongodb是顺序的，不是哈希表，所以不能实现redis存储结构2类似的存储结构

# Neo4j

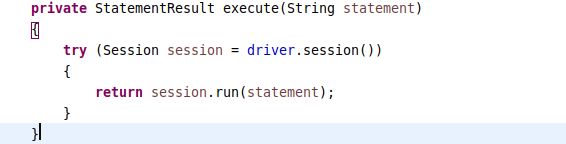
## 实验步骤

### 存储结构

将数据作为一个图。图上的点的标签为subject或object，有属性name。每个三元组是图上的一条边，predicate是这条边的属性，这条边从subject指向object。如果subject这个点不存在，则增加这个点。

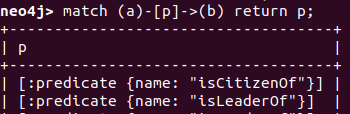






点和边如图所示：



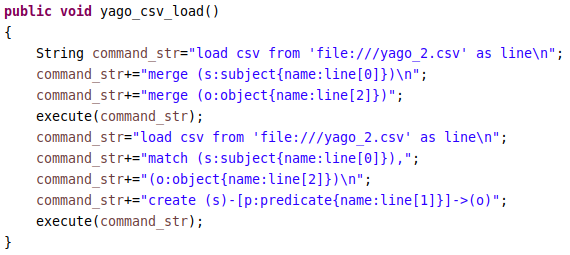


由于neo4j创建点和边的速度较慢，所以只取了一部分数据

### LOAD CSV

改进数据集的导入，用load csv方法，该方法适合百万量级的数据，速度较快，但是由于内存不足，所以仍只取一部分数据。

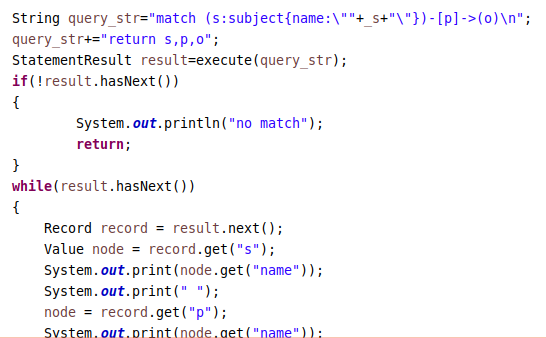
创建节点使用merge避免重复创建.



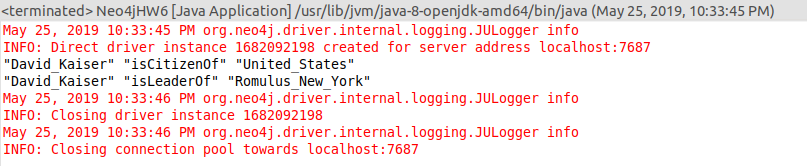
### 查询1

给定一个si，给出它所有的P和O，<si, P, O>

查询所有s为si的关系,即为结果



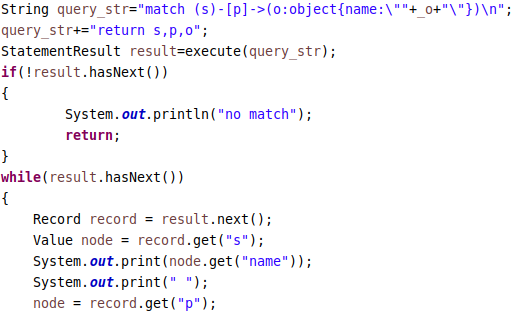
neo\_obj.query\_1("David\_Kaiser");



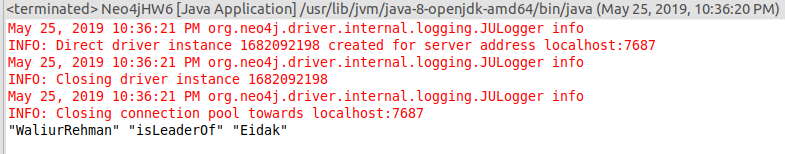
### 查询2

给定一个oi, 给出它所有的S和P，<S, P,oi>

查询所有宾语为oi的关系



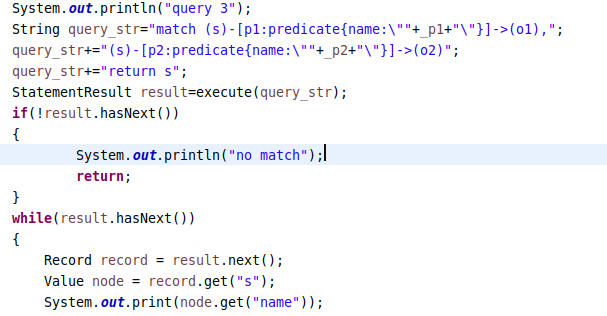
neo\_obj.query\_2("Eidak");



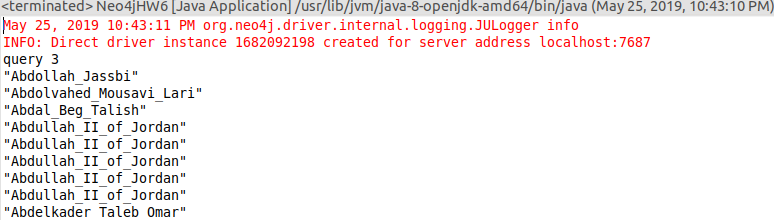
### 查询3

给定两个p1,p2, 给出同时拥有它们的S，<S, p1, \*>, <S, p2, \*>

匹配节点s,s连有p1,p2两条出边,符合这种图形的s即为结果.又因为s是图中的节点且创建节点时使用merge,所以s没有重复.



neo\_obj.query\_3("isLeaderOf","isCitizenOf");



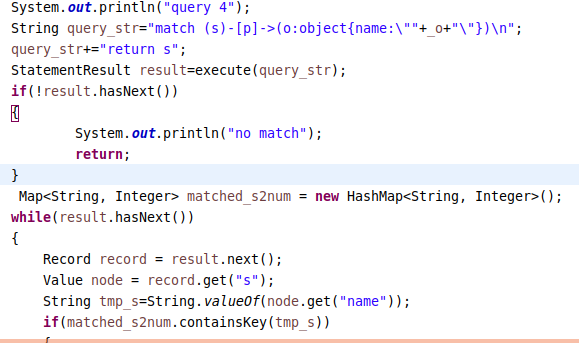
### 查询4

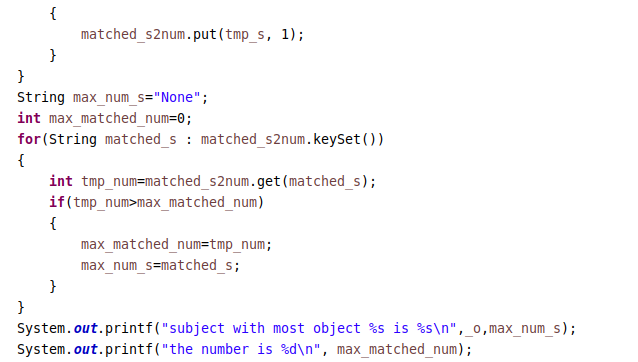
给定一个oi, 给出拥有这样oi最多的S

Map<String, Integer> matched\_s2num = new HashMap<String, Integer>();记录每个s匹配的次数

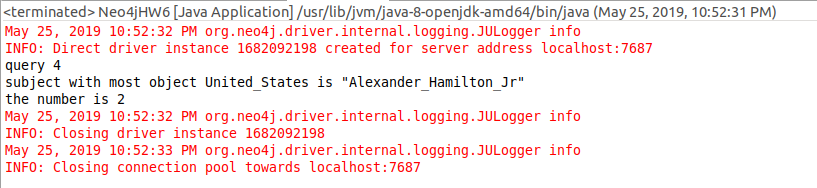
查找所有宾语为oi的关系,更新该关系中s匹配的计数.

计数最高的s即为答案



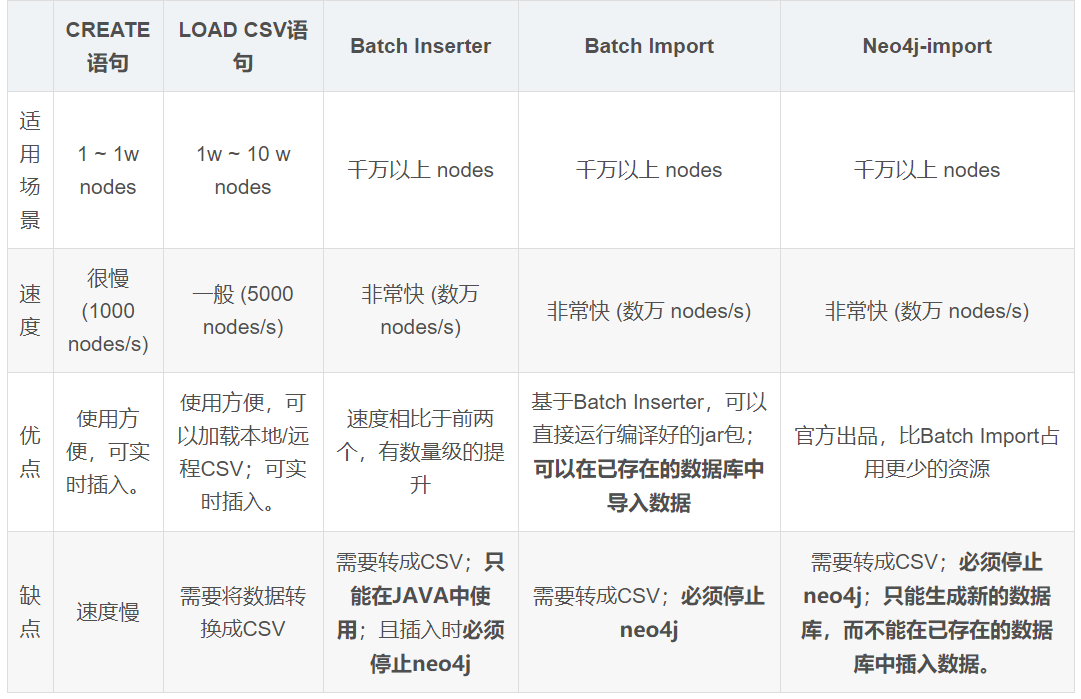


neo\_obj.query\_4("United\_States");



## 心得

neo4j逐个创建点和关系的速度较慢，可将数据转换csv后用load csv批量导入



load csv 导入的csv文件需要放在neo4j的import文件夹下

neo4j 的图结构最适合三元组的语义

# 三者比较

neo4j 的图数据库,有节点和关系,最适合三元组的语义,主语和宾语为节点,而谓语是主语指向宾语的边.

neo4j创建节点和关系的速度最慢,因为要构建图.

mongodb创建和检索最快,虽然只能读取所有文档,但因为是顺序检索所以访存开销小

redis创建和检索数据比mongodb慢,因为哈希表并不连续存储,所以访存开销较大

# Appendix

## Reddis

### load\_triples

public static void load\_triples(String triples\_path)

{

try

{

//连接本地Redis服务

Jedis jedis = new Jedis("localhost");

System.out.println("连接成功");

//查看服务运行状态

System.out.println("服务正在运行: "+jedis.ping());

System.out.println("read triples into redis hashtable");

BufferedReader br = new BufferedReader(new FileReader(triples\_path));

// ArrayList<String> cols=new ArrayList<String>();

int line\_cnt=0;

String line\_str=null;

String[] cols= {"s","p","o"};

while(true)

{

line\_str=br.readLine();

if(line\_str==null)

break;

String[] items=line\_str.split(" ");

int item\_num=items.length;

if(item\_num<1)

break;

for(int col\_i=0;col\_i<3;++col\_i)

{

items[col\_i]=items[col\_i].trim();

}

// write into hashtable

Map<String,String> line\_triple=new HashMap<String, String>();

for(int col\_i=0;col\_i<3;++col\_i)

{

line\_triple.put(cols[col\_i], items[col\_i]);

}

jedis.hmset(String.valueOf(line\_cnt), line\_triple);

++line\_cnt;

}

}

### 查询1

public static void query\_1(String \_s)

{

int line\_num=1044;

//连接本地Redis服务

Jedis jedis = new Jedis("localhost");

System.out.println("连接成功");

//查看服务运行状态

System.out.println("服务正在运行: "+jedis.ping());

System.out.println("query 1");

int matched\_triple\_num=0;

try

{

for(int id=0;id<triple\_num;++id)

{

String key=String.valueOf(id);

String tmp\_s=jedis.hget(key, "s");

if(tmp\_s.equals(\_s))

{

// this triple matched

String tmp\_p=jedis.hget(key, "p");

String tmp\_o=jedis.hget(key, "o");

String output\_str="<"+tmp\_s+","+tmp\_p+","+tmp\_o+">";

System.out.println(output\_str);

++matched\_triple\_num;

}

}

System.out.println("matched triple num: "+String.valueOf(matched\_triple\_num));

}

catch(Exception e)

{

e.printStackTrace();

}

}

### 查询2

public static void query\_2(String \_o)

{

//连接本地Redis服务

Jedis jedis = new Jedis("localhost");

System.out.println("连接成功");

//查看服务运行状态

System.out.println("服务正在运行: "+jedis.ping());

String query\_pattern="<?s,?p,"+\_o+">";

System.out.println("query 2: "+query\_pattern);

int matched\_triple\_num=0;

try

{

for(int id=0;id<triple\_num;++id)

{

String key=String.valueOf(id);

String tmp\_o=jedis.hget(key, "o");

if(tmp\_o.equals(\_o))

{

// this triple matched

String tmp\_s=jedis.hget(key, "s");

String tmp\_p=jedis.hget(key, "p");

String output\_str="<"+tmp\_s+","+tmp\_p+","+tmp\_o+">";

System.out.println(output\_str);

++matched\_triple\_num;

}

}

System.out.println("matched triple num: "+String.valueOf(matched\_triple\_num));

}

catch(Exception e)

{

e.printStackTrace();

}

}

### 查询3

public static void query\_3(String \_p1,String \_p2)

{

//连接本地Redis服务

Jedis jedis = new Jedis("localhost");

System.out.println("连接成功");

//查看服务运行状态

System.out.println("服务正在运行: "+jedis.ping());

String query\_pattern="<?s,"+\_p1+",?o> "+"<?s,"+\_p2+",?o> ";

System.out.println("query 2: "+query\_pattern);

ArrayList<String> array\_p1=new ArrayList<String>();

ArrayList<String> array\_p2=new ArrayList<String>();

try

{

for(int id=0;id<triple\_num;++id)

{

String key=String.valueOf(id);

String tmp\_p=jedis.hget(key, "p");

if(tmp\_p.equals(\_p1))

{

String tmp\_s=jedis.hget(key, "s");

array\_p1.add(tmp\_s);

}

if(tmp\_p.equals(\_p2))

{

String tmp\_s=jedis.hget(key, "s");

array\_p2.add(tmp\_s);

}

}

String[] slist\_1=new String[]{};

slist\_1=(String[])array\_p1.toArray(slist\_1);

String[] slist\_2=new String[]{};

slist\_2=(String[])array\_p2.toArray(slist\_2);

Arrays.sort(slist\_1);

Arrays.sort(slist\_2);

// remove duplicated element

// union

int len1=slist\_1.length;

int len2=slist\_2.length;

int id1=0;

int id2=0;

int matched\_s\_num=0;

while(id1<len1&&id2<len2)

{

String s1=slist\_1[id1];

String s2=slist\_2[id2];

if(s1.equals(s2))

{

System.out.println(s1);

++matched\_s\_num;

++id1;

++id2;

}

else if(s1.compareTo(s2)<0)

{

++id1;

}

else

{

++id2;

}

}

System.out.println(String.valueOf(matched\_s\_num)+" matched subject");

}

catch(Exception e)

{

e.printStackTrace();

}

}

### 存储结构2

public static void struct\_q3(String \_triples\_path)

{

try

{

Jedis jedis = new Jedis("localhost");

System.out.println("连接成功");

System.out.println("服务正在运行: "+jedis.ping());

System.out.println("read triples into redis hashtable");

BufferedReader br = new BufferedReader(new FileReader(\_triples\_path));

String line\_str=null;

while(true)

{

line\_str=br.readLine();

if(line\_str==null)

break;

String[] items=line\_str.split(" ");

int item\_num=items.length;

if(item\_num<1)

break;

for(int col\_i=0;col\_i<3;++col\_i)

{

items[col\_i]=items[col\_i].trim();

}

// write into hashtable

Map<String,String> p2o=new HashMap<String, String>();

p2o.put(items[1],items[2]);

jedis.hmset(items[0], p2o);

}

}

catch(Exception e)

{

e.printStackTrace();

}

}

### 查询4

public static void query\_4(String \_o)

{

try

{

Jedis jedis = new Jedis("localhost");

System.out.println("连接成功");

System.out.println("服务正在运行: "+jedis.ping());

System.out.println("query 4");

Map<String, Integer> matched\_s2num = new HashMap<String, Integer>();

for(int id=0;id<triple\_num;++id)

{

String key=String.valueOf(id);

String tmp\_o=jedis.hget(key,"o");

if(tmp\_o.equals(\_o))

{

String tmp\_s=jedis.hget(key, "s");

// count += 1

if(matched\_s2num.containsKey(tmp\_s))

{

int prev\_cnt=matched\_s2num.get(tmp\_s);

matched\_s2num.put(tmp\_s, prev\_cnt+1);

}

else

{

matched\_s2num.put(tmp\_s, 1);

}

}

}

String max\_num\_s="None";

int max\_matched\_num=0;

for(String matched\_s : matched\_s2num.keySet())

{

int tmp\_num=matched\_s2num.get(matched\_s);

if(tmp\_num>max\_matched\_num)

{

max\_matched\_num=tmp\_num;

max\_num\_s=matched\_s;

}

}

System.out.printf("subject with most object %s is %s\n",\_o,max\_num\_s);

System.out.printf("the number is %d\n", max\_matched\_num);

}

catch(Exception e)

{

e.printStackTrace();

}

}

## MongoDB

### 存储结构

public static void load\_triples(String triples\_path)

{

try

{

MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

MongoDatabase mongoDatabase = mongoClient.getDatabase("test");

mongoDatabase.createCollection("yago");

MongoCollection<Document> collection = mongoDatabase.getCollection("yago");

List<Document> documents = new ArrayList<Document>();

BufferedReader br = new BufferedReader(new FileReader(triples\_path));

// ArrayList<String> cols=new ArrayList<String>();

String line\_str=null;

String[] cols= {"s","p","o"};

while(true)

{

line\_str=br.readLine();

if(line\_str==null)

break;

String[] items=line\_str.split(" ");

int item\_num=items.length;

if(item\_num<1)

break;

for(int col\_i=0;col\_i<3;++col\_i)

{

items[col\_i]=items[col\_i].trim();

}

Document document=new Document();

for(int item\_i=0;item\_i<3;++item\_i)

{

document.append(cols[item\_i], items[item\_i]);

}

documents.add(document);

}

collection.insertMany(documents);

}

catch(Exception e)

{

e.printStackTrace();

}

}

### 查询1

public static void query\_1(String \_s)

{

try

{

MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

MongoDatabase mongoDatabase = mongoClient.getDatabase("test");

MongoCollection<Document> collection = mongoDatabase.getCollection("yago");

FindIterable<Document> findIterable = collection.find();

MongoCursor<Document> mongoCursor = findIterable.iterator();

int matched\_triple\_num=0;

while(mongoCursor.hasNext())

{

Document doc=mongoCursor.next();

String tmp\_s=(String)doc.get("s");

if(tmp\_s.equals(\_s))

{

// matched

String tmp\_p=(String)doc.get("p");

String tmp\_o=(String)doc.get("o");

System.out.printf("<%s,%s,%s>\n",tmp\_s,tmp\_p,tmp\_o);

++matched\_triple\_num;

}

}

System.out.println("matched triple num:"+String.valueOf(matched\_triple\_num));

}

catch(Exception e)

{

e.printStackTrace();

}

}

### 查询2

public static void query\_2(String \_o)

{

try

{

System.out.println("query 2");

MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

MongoDatabase mongoDatabase = mongoClient.getDatabase("test");

MongoCollection<Document> collection = mongoDatabase.getCollection("yago");

FindIterable<Document> findIterable = collection.find();

MongoCursor<Document> mongoCursor = findIterable.iterator();

int matched\_triple\_num=0;

while(mongoCursor.hasNext())

{

Document doc=mongoCursor.next();

String tmp\_o=(String)doc.get("o");

if(tmp\_o.equals(\_o))

{

// matched

String tmp\_s=(String)doc.get("s");

String tmp\_p=(String)doc.get("p");

System.out.printf("<%s,%s,%s>\n",tmp\_s,tmp\_p,tmp\_o);

++matched\_triple\_num;

}

}

System.out.println("matched triple num:"+String.valueOf(matched\_triple\_num));

}

catch(Exception e)

{

e.printStackTrace();

}

}

### 查询3

public static void query\_3(String \_p1,String \_p2)

{

try

{

System.out.println("query 3");

MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

MongoDatabase mongoDatabase = mongoClient.getDatabase("test");

MongoCollection<Document> collection = mongoDatabase.getCollection("yago");

FindIterable<Document> findIterable = collection.find();

MongoCursor<Document> mongoCursor = findIterable.iterator();

ArrayList<String> s\_array\_1=new ArrayList<String>();

ArrayList<String> s\_array\_2=new ArrayList<String>();

while(mongoCursor.hasNext())

{

Document doc=mongoCursor.next();

String tmp\_p=(String)doc.get("p");

if(tmp\_p.equals(\_p1))

{

String tmp\_s=(String)doc.get("s");

s\_array\_1.add(tmp\_s);

}

if(tmp\_p.equals(\_p2))

{

String tmp\_s=(String)doc.get("s");

s\_array\_2.add(tmp\_s);

}

}

String[] slist\_1=new String[]{};

slist\_1=(String[])s\_array\_1.toArray(slist\_1);

String[] slist\_2=new String[]{};

slist\_2=(String[])s\_array\_2.toArray(slist\_2);

Arrays.sort(slist\_1);

Arrays.sort(slist\_2);

int len\_prev\_1=slist\_1.length;

int len\_prev\_2=slist\_2.length;

ArrayList<String> sarray\_1=new ArrayList<String>();

ArrayList<String> sarray\_2=new ArrayList<String>();

sarray\_1.add(slist\_1[0]);

for(int i=1;i<len\_prev\_1;++i)

{

if(slist\_1[i].equals(slist\_1[i-1]))

continue;

sarray\_1.add(slist\_1[i]);

}

sarray\_2.add(slist\_2[0]);

for(int i=1;i<len\_prev\_2;++i)

{

if(slist\_2[i].equals(slist\_2[i-1]))

continue;

sarray\_2.add(slist\_2[i]);

}

slist\_1=(String[])sarray\_1.toArray(slist\_1);

slist\_2=(String[])sarray\_2.toArray(slist\_2);

int len1=slist\_1.length;

int len2=slist\_2.length;

int id1=0;

int id2=0;

int matched\_s\_num=0;

while(id1<len1&&id2<len2)

{

String s1=slist\_1[id1];

String s2=slist\_2[id2];

if(s1==null)

break;

if(s2==null)

break;

if(s1.equals(s2))

{

System.out.println(s1);

++matched\_s\_num;

++id1;

++id2;

}

else if(s1.compareTo(s2)<0)

{

++id1;

}

else

{

++id2;

}

}

System.out.println(String.valueOf(matched\_s\_num)+" matched subject");

}

catch(Exception e)

{

e.printStackTrace();

}

}

### 查询4

public static void query\_4(String \_o)

{

try

{

System.out.println("query 4");

MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

MongoDatabase mongoDatabase = mongoClient.getDatabase("test");

MongoCollection<Document> collection = mongoDatabase.getCollection("yago");

FindIterable<Document> findIterable = collection.find();

MongoCursor<Document> mongoCursor = findIterable.iterator();

Map<String, Integer> matched\_s2num = new HashMap<String, Integer>();

while(mongoCursor.hasNext())

{

Document doc=mongoCursor.next();

String tmp\_o=(String)doc.get("o");

if(tmp\_o.equals(\_o))

{

String tmp\_s=(String)doc.get("s");

// count += 1

if(matched\_s2num.containsKey(tmp\_s))

{

int prev\_cnt=matched\_s2num.get(tmp\_s);

matched\_s2num.put(tmp\_s, prev\_cnt+1);

}

else

{

matched\_s2num.put(tmp\_s, 1);

}

}

}

String max\_num\_s="None";

int max\_matched\_num=0;

for(String matched\_s : matched\_s2num.keySet())

{

int tmp\_num=matched\_s2num.get(matched\_s);

if(tmp\_num>max\_matched\_num)

{

max\_matched\_num=tmp\_num;

max\_num\_s=matched\_s;

}

}

System.out.printf("subject with most object %s is %s\n",\_o,max\_num\_s);

System.out.printf("the number is %d\n", max\_matched\_num);

}

catch(Exception e)

{

e.printStackTrace();

}

}

## Neo4j

### 存储结构

public void load\_triples(String \_triples\_path)

{

try

{

BufferedReader br = new BufferedReader(new FileReader(\_triples\_path));

String line\_str=null;

int line\_cnt=0;

while(true)

{

line\_str=br.readLine();

if(line\_str==null)

break;

String[] items=line\_str.split(" ");

int item\_num=items.length;

if(item\_num<1)

break;

for(int col\_i=0;col\_i<3;++col\_i)

{

items[col\_i]=items[col\_i].trim();

}

String tmp\_s=items[0];

String tmp\_p=items[1];

String tmp\_o=items[2];

String command\_find\_s="match (s:subject{name:\""+tmp\_s+"\"}) return s";

StatementResult result = execute(command\_find\_s);

boolean s\_found;

if(!result.hasNext())

s\_found=false;

else

s\_found=true;

if(!s\_found)

{

String command\_create\_s="create (s:subject{name:\""+tmp\_s+"\"})";

execute(command\_create\_s);

}

String command\_find\_o="match (o:object{name:\""+tmp\_o+"\"}) return o";

StatementResult result\_o = execute(command\_find\_o);

boolean o\_found;

if(!result.hasNext())

o\_found=false;

else

o\_found=true;

if(!o\_found)

{

String command\_create\_o="create (o:object{name:\""+tmp\_o+"\"})";

execute(command\_create\_o);

}

String command\_create\_p="match (s:subject{name:\""+tmp\_s+"\"})"

+",(o:object{name:\""+tmp\_o+"\"}) ";

command\_create\_p+="create (s)-[p:predicate{name:\""+tmp\_p+"\"}]->(o)";

execute(command\_create\_p);

++line\_cnt;

if(line\_cnt%100==0)

System.out.printf("have read %d lines\n",line\_cnt);

}

}

catch(Exception e)

{

e.printStackTrace();

}

}

### LOAD CSV

public void yago\_csv\_load()

{

String command\_str="load csv from 'file:///yago\_2.csv' as line\n";

command\_str+="merge (s:subject{name:line[0]})\n";

command\_str+="merge (o:object{name:line[2]})";

execute(command\_str);

command\_str="load csv from 'file:///yago\_2.csv' as line\n";

command\_str+="match (s:subject{name:line[0]}),";

command\_str+="(o:object{name:line[2]})\n";

command\_str+="create (s)-[p:predicate{name:line[1]}]->(o)";

execute(command\_str);

}

### 查询1

public void query\_1(String \_s)

{

try

{

String query\_str="match (s:subject{name:\""+\_s+"\"})-[p]->(o)\n";

query\_str+="return s,p,o";

StatementResult result=execute(query\_str);

if(!result.hasNext())

{

System.out.println("no match");

return;

}

while(result.hasNext())

{

Record record = result.next();

Value node = record.get("s");

System.out.print(node.get("name"));

System.out.print(" ");

node = record.get("p");

System.out.print(node.get("name"));

System.out.print(" ");

node = record.get("o");

System.out.print(node.get("name"));

System.out.println(" ");

}

}

catch(Exception e)

{

e.printStackTrace();

}

}

### 查询2

public void query\_2(String \_o)

{

try

{

String query\_str="match (s)-[p]->(o:object{name:\""+\_o+"\"})\n";

query\_str+="return s,p,o";

StatementResult result=execute(query\_str);

if(!result.hasNext())

{

System.out.println("no match");

return;

}

while(result.hasNext())

{

Record record = result.next();

Value node = record.get("s");

System.out.print(node.get("name"));

System.out.print(" ");

node = record.get("p");

System.out.print(node.get("name"));

System.out.print(" ");

node = record.get("o");

System.out.print(node.get("name"));

System.out.println(" ");

}

}

catch(Exception e)

{

e.printStackTrace();

}

}

### 查询3

public void query\_3(String \_p1,String \_p2)

{

try

{

System.out.println("query 3");

String query\_str="match (s)-[p1:predicate{name:\""+\_p1+"\"}]->(o1),";

query\_str+="(s)-[p2:predicate{name:\""+\_p2+"\"}]->(o2)";

query\_str+="return s";

StatementResult result=execute(query\_str);

if(!result.hasNext())

{

System.out.println("no match");

return;

}

while(result.hasNext())

{

Record record = result.next();

Value node = record.get("s");

System.out.print(node.get("name"));

System.out.println(" ");

}

}

catch(Exception e)

{

e.printStackTrace();

}

}

### 查询4

public void query\_4(String \_o)

{

try

{

System.out.println("query 4");

String query\_str="match (s)-[p]->(o:object{name:\""+\_o+"\"})\n";

query\_str+="return s";

StatementResult result=execute(query\_str);

if(!result.hasNext())

{

System.out.println("no match");

return;

}

Map<String, Integer> matched\_s2num = new HashMap<String, Integer>();

while(result.hasNext())

{

Record record = result.next();

Value node = record.get("s");

String tmp\_s=String.valueOf(node.get("name"));

if(matched\_s2num.containsKey(tmp\_s))

{

int prev\_cnt=matched\_s2num.get(tmp\_s);

matched\_s2num.put(tmp\_s, prev\_cnt+1);

}

else

{

matched\_s2num.put(tmp\_s, 1);

}

}

String max\_num\_s="None";

int max\_matched\_num=0;

for(String matched\_s : matched\_s2num.keySet())

{

int tmp\_num=matched\_s2num.get(matched\_s);

if(tmp\_num>max\_matched\_num)

{

max\_matched\_num=tmp\_num;

max\_num\_s=matched\_s;

}

}

System.out.printf("subject with most object %s is %s\n",\_o,max\_num\_s);

System.out.printf("the number is %d\n", max\_matched\_num);

}

catch(Exception e)

{

e.printStackTrace();

}

}