neo4j-java11

sudo add-apt-repository ppa:openjdk-r/ppa

sudo apt-get update

sudo apt-get install openjdk-11-jdk

从这开始

java -version

JAVA\_HOME="/usr/lib/jvm/java-11-openjdk-amd64/"

source /etc/environment

echo $JAVA\_HOME

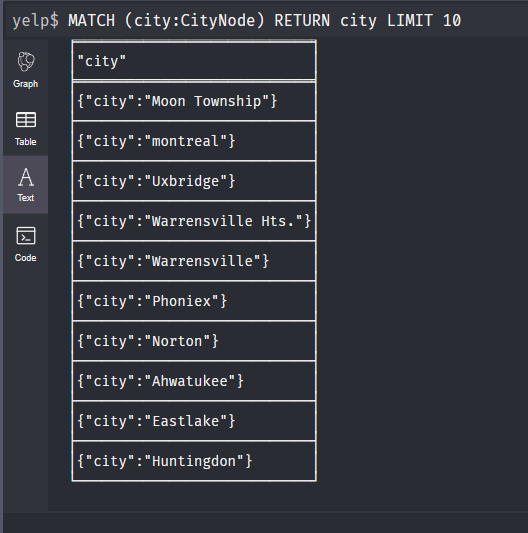
./neo4j-admin import --id-type=STRING --database=yelp --nodes=UserNode="../import/user\_header.csv,../import/user.csv" --nodes=ReviewNode="../import/review\_header.csv,../import/review.csv" --nodes=BusinessNode="../import/business\_header.csv,../import/BusinessNode.csv" --nodes=CityNode="../import/city\_header.csv,../import/CityNode.csv" --nodes=CategoryNode="../import/category\_header.csv,../import/CategoryNode.csv" --relationships=HasFriend="../import/user\_FRIENDS\_user\_header.csv,../import/user\_FRIENDS\_user.csv" --relationships=Review="../import/user\_WROTE\_review\_header.csv,../import/user\_WROTE\_review.csv" --relationships=Reviewed="../import/review\_REVIEWS\_business\_header.csv,../import/review\_REVIEWS\_business.csv" --relationships=IN\_CITY="../import/business\_IN\_CITY\_city\_header.csv,../import/business\_IN\_CITY\_city.csv" --relationships=IN\_CATEGORY="../import/business\_IN\_CATEGORY\_category\_header.csv,../import/business\_IN\_CATEGORY\_category.csv" --multiline-fields=true --skip-bad-relationships

使用cd neo4j-community-4.0.9/bin

启动neo4j：进入解压后的文件夹的 bin 目录下，使用./neo4j console 或./neo4j start 命令即可启动数据库

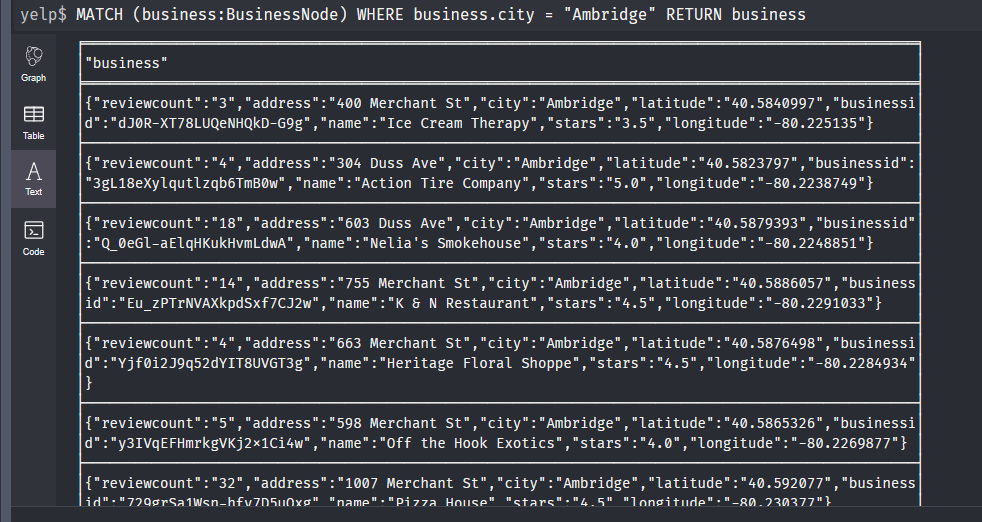
1.查询标签是 CityNode 的节点，限制 10 个

MATCH (city:CityNode) RETURN city LIMIT 10



2. 查询城市是 Ambridge 的商家节点。

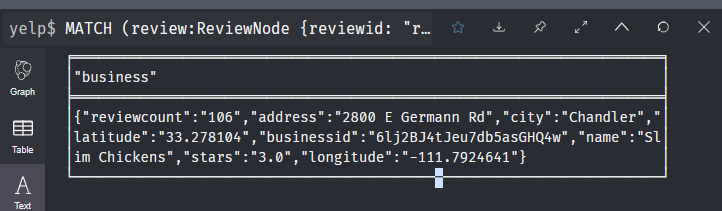
MATCH (business:BusinessNode) WHERE business.city = "Ambridge" RETURN business



3. 查询reviewid是rEITo90tpyKmEfNDp3Ou3A对应的bussiness信息。

MATCH (review:ReviewNode {reviewid: "rEITo90tpyKmEfNDp3Ou3A"})-[:REVIEWED]->(business:BusinessNode)

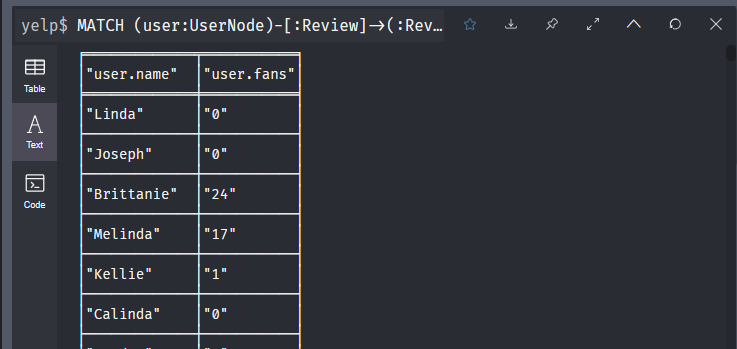
RETURN business



4. 查询评价过businessid是fyJAqmweGm8VXnpU4CWGNw商家的用户的名字和 粉丝数。

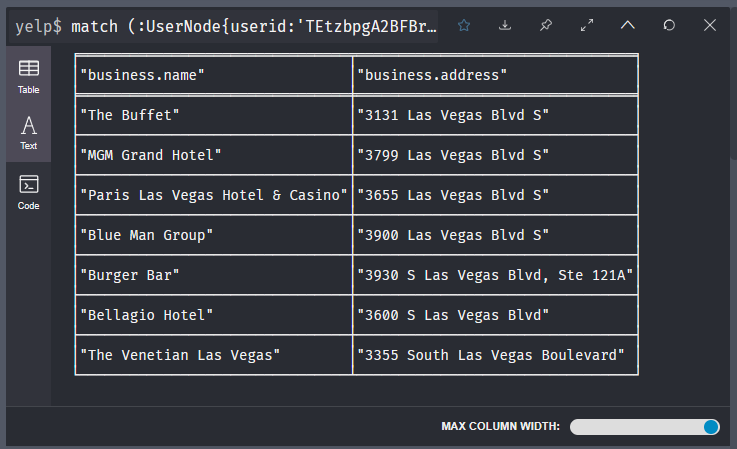
MATCH (user:UserNode)-[:Review]->(:ReviewNode)-[:Reviewed]->(:BusinessNode {businessid: 'fyJAqmweGm8VXnpU4CWGNw'})

RETURN user.name, user.fans



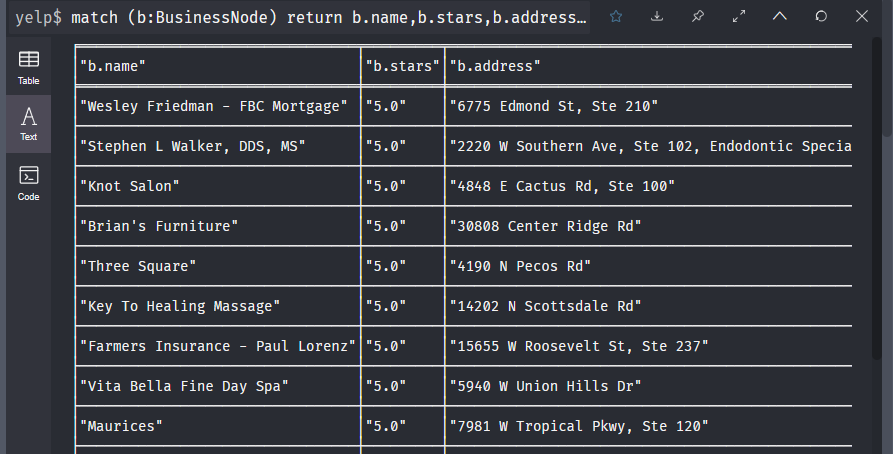
5. 查询被userid为TEtzbpgA2BFBrC0y0sCbfw的用户评论为5星的商家名称和地址。

match (:UserNode{userid:'TEtzbpgA2BFBrC0y0sCbfw'})-[:Review]->(:ReviewNode{stars:'5.0'})-[:Reviewed]->(business:BusinessNode) return business.name,business.address



6. 查询商家名及对应的星级和地址，按照星级降序排序（限制15条）。

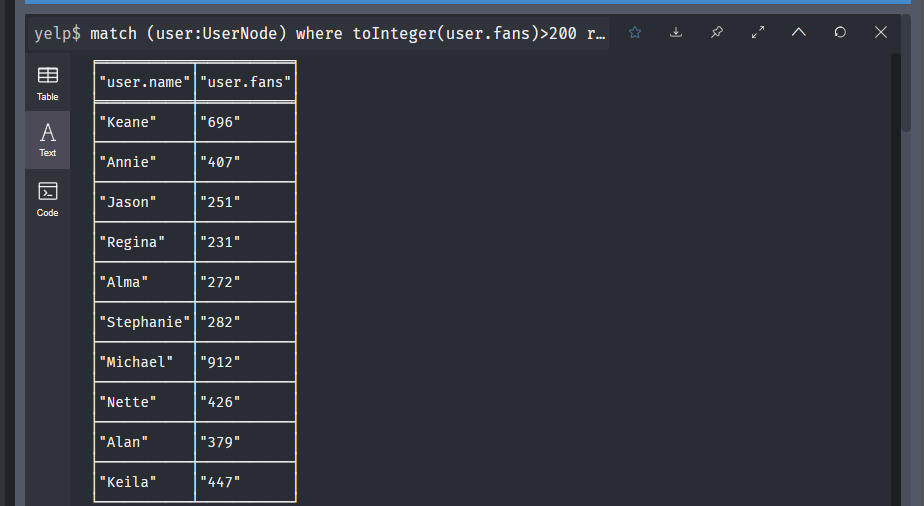
match (b:BusinessNode) return b.name,b.stars,b.address order by b.stars desc limit 15



7.使用where查询粉丝数大于200的用户的名字和粉丝数（限制10条）。

match (user:UserNode) where toInteger(user.fans)>200 return user.name,user.fans limit 10

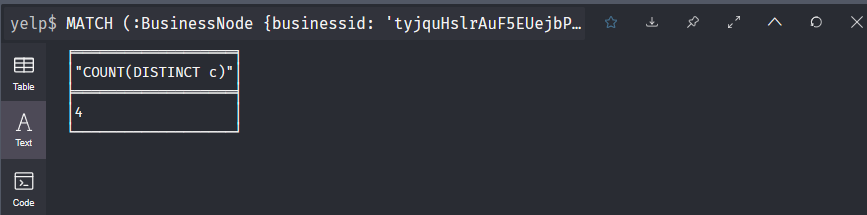
toInteger转换为整型数据



8.查询businessid是tyjquHslrAuF5EUejbPfrw商家包含的种类数,并使用 PROFILE查看执行计划，进行说明。

match (:BusinessNode {businessid: 'tyjquHslrAuF5EUejbPfrw'})-[:IN\_CATEGORY]->(c:CategoryNode)

return count(c)



Profile match (:BusinessNode {businessid: 'tyjquHslrAuF5EUejbPfrw'})-[:IN\_CATEGORY]->(c:CategoryNode)

return count(c)

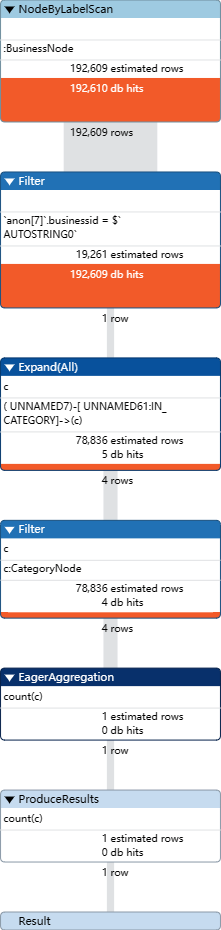
使用 PROFILE 可以获取以下信息：

查询执行计划：你将看到Neo4j的查询执行计划，包括节点的访问方式、关系的遍历方式以及任何索引的使用。

访问路径的成本估算：Neo4j将提供每个访问路径的成本估算，以帮助你了解哪些部分的查询比较耗时。

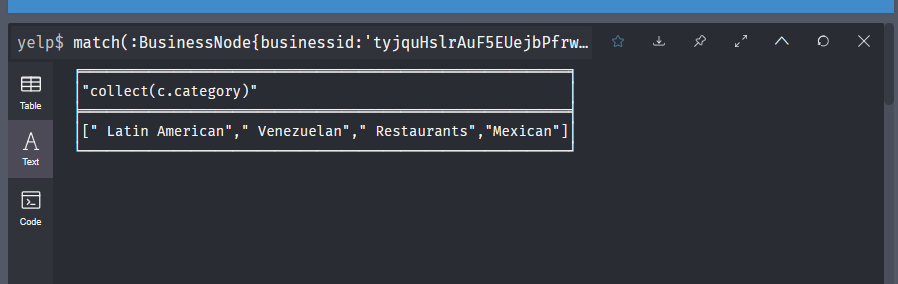
节点和关系的统计信息：你将获得有关节点和关系的统计信息，例如访问的次数、处理的数据量等。

执行时间：你可以查看查询的实际执行时间，以帮助识别性能瓶颈。



9.查询businessid是tyjquHslrAuF5EUejbPfrw商家包含的种类,以list的形式返 回。

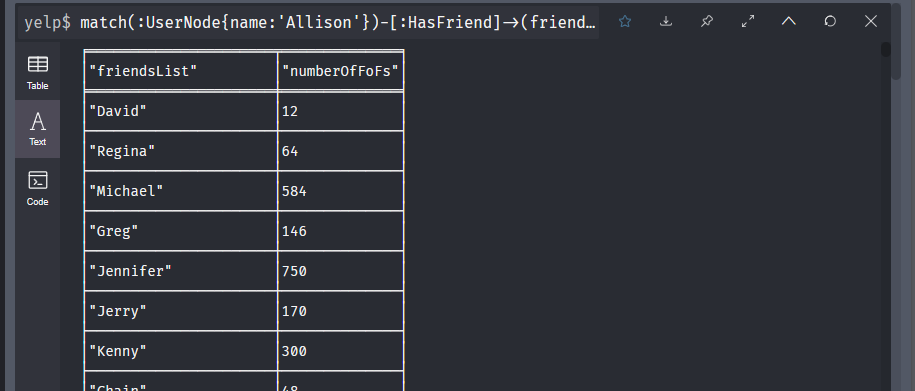
match(:BusinessNode{businessid:'tyjquHslrAuF5EUejbPfrw'})-[:IN\_CATEGORY]->(c:CategoryNode) return collect(c.category)



10.查询Allison的朋友（直接相邻）分别有多少位朋友。(考察：使用with传递查询 结果到后续的处理)

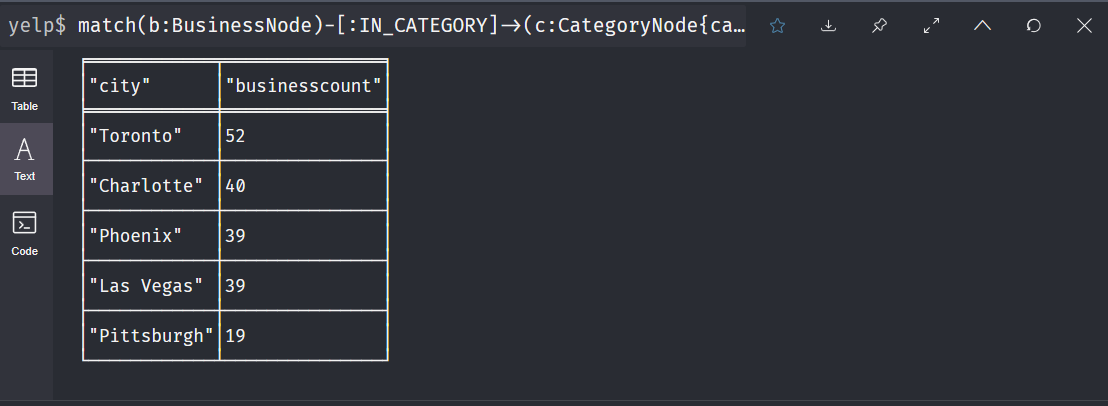
match(:UserNode{name:'Allison'})-[:HasFriend]->(friend) with friend.name as friendsList,size((friend)-[:HasFriend]-()) as numberOfFoFs return friendsList,numberOfFoFs

with关键字可以进一步处理结果



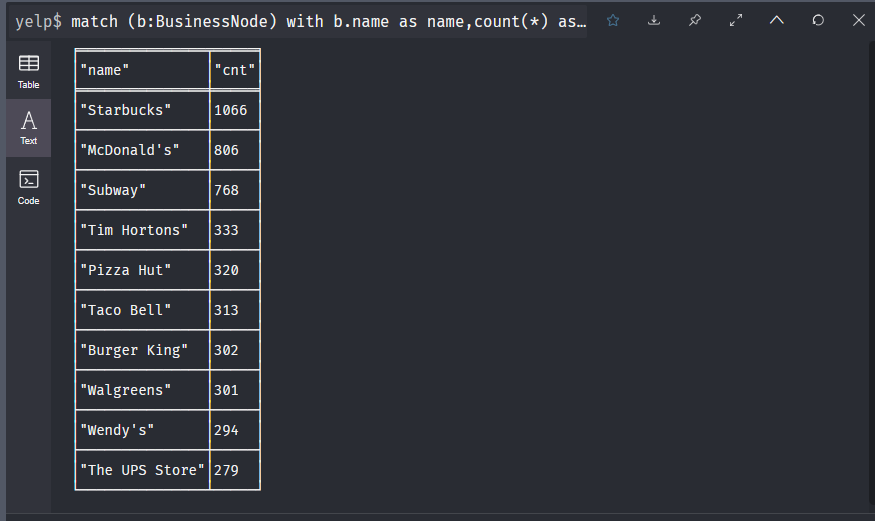
11.查询拥有类别为Salad的商家数量前5的城市，返回城市名称和商家数量。

match(b:BusinessNode)-[:IN\_CATEGORY]->(c:CategoryNode{category:'Salad'}) with b, b.city as city return city,count(b) as businesscount order by businesscount desc limit 5



12.查询商家名重复次数前10的商家名及其次数。

match (b:BusinessNode) with b.name as name,count(\*) as cnt where cnt>1 return name,cnt order by cnt desc limit 10



13. 统计评价数大于5000的商家名热度（名字的重复的次数在所有的商家名中的占 比），按照评价数量排序，返回热度和商家名和评价数。

MATCH (business:BusinessNode)

WHERE toInteger(business.reviewcount) > 5000

WITH COUNT(distinct business) AS cnt

MATCH (business:BusinessNode)

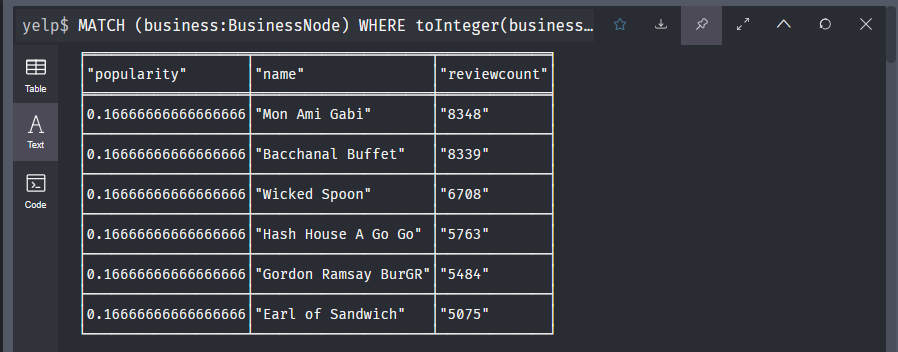
WHERE toInteger(business.reviewcount) > 5000

WITH business, COUNT(\*) AS count, cnt

WITH business.name AS name, count, count\*1.0/cnt AS popularity, business.reviewcount AS reviewcount

RETURN popularity, name, reviewcount

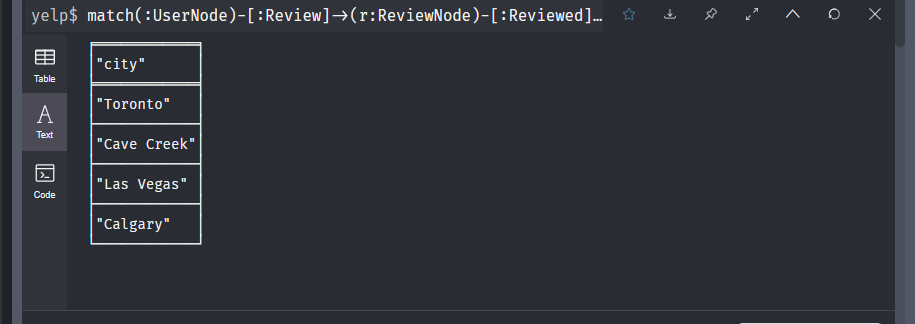
ORDER BY reviewcount DESC



14. 查询具有评分为5.0的Zoos类别的商铺所在城市。

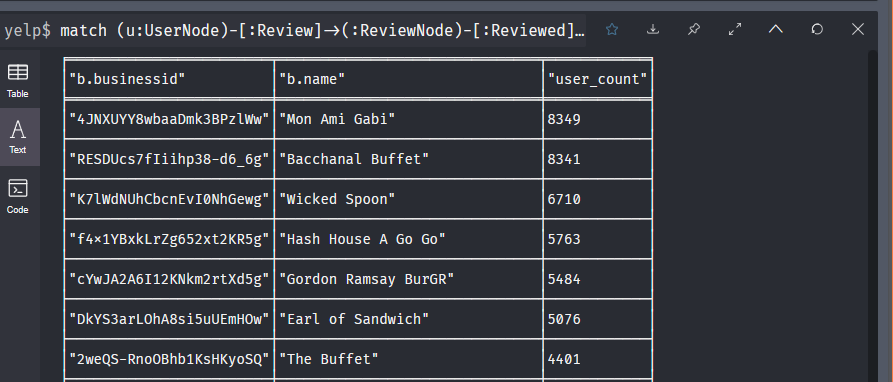
match(:UserNode)-[:Review]->(r:ReviewNode)-[:Reviewed]->(b:BusinessNode)-[:IN\_CATEGORY]->(c:CategoryNode{category:'Zoos'}) where r.stars = '5.0' return distinct b.city as city

每个城市会有多个5.0分的商铺 因此只输出唯一的城市名字



15.统计每个商家被多少个不同用户评论过，按照此数量降序排列，返回商家id，商 家名和此商家被多少个不同用户评论过，结果限制10条记录。

match (u:UserNode)-[:Review]->(:ReviewNode)-[:Reviewed]->(b:BusinessNode) with b,count(distinct u) as user\_count return b.businessid,b.name,user\_count order by user\_count desc limit 10



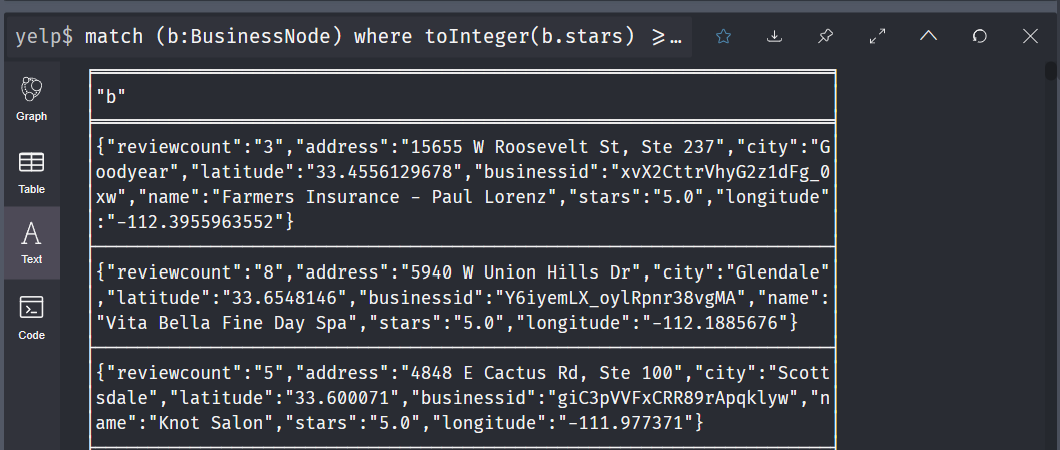
16. 体会建立索引对查询带来的性能提升，但会导致插入，删除等操作变慢（需要额外维护索引代价）。

**为建立索引前：**

查询所有星级大于4.5的商铺：

match (b:BusinessNode) where toInteger(b.stars) >= 4.5 return b

**消耗**Started streaming 28216 records after 1 ms and completed after 8 ms, displaying first 1000 rows.



为这些商铺添加flag标识，标记他们是优良商铺：

match (b:BusinessNode) where toInteger(b.stars) >= 4.5 set b.flag = 1

**消耗：**Set 28216 properties, completed after 1310 ms.

删除添加的标记

match (b:BusinessNode) where toInteger(b.stars) >= 4.5 remove b.flag

**消耗：**Set 28216 properties, completed after 1164 ms.

**为建立索引后：**

重新添加flag

添加索引：create index for(user:UserNode) on (user.flag)

重新删除索引：

match (b:BusinessNode) where toInteger(b.stars) >= 4.5 remove b.flag

Set 28216 properties, completed after 986 ms.

**有提升！**

**删除索引：**

call db.indexes

drop index index\_7494c4a

17. 查询与用户 user1 （ userid: tvZKPah2u9G9dFBg5GT0eg ) 不是朋友关 系的用户中和 user1 评价过相同的商家的用户，返回用户名、共同评价的商家的数 量，按照评价数量降序排序（查看该查询计划，并尝试根据查询计划优化）。

match (user1:UserNode {userid: 'tvZKPah2u9G9dFBg5GT0eg'})

-[:Review]->(:ReviewNode)-[:Reviewed]->(b:BusinessNode)

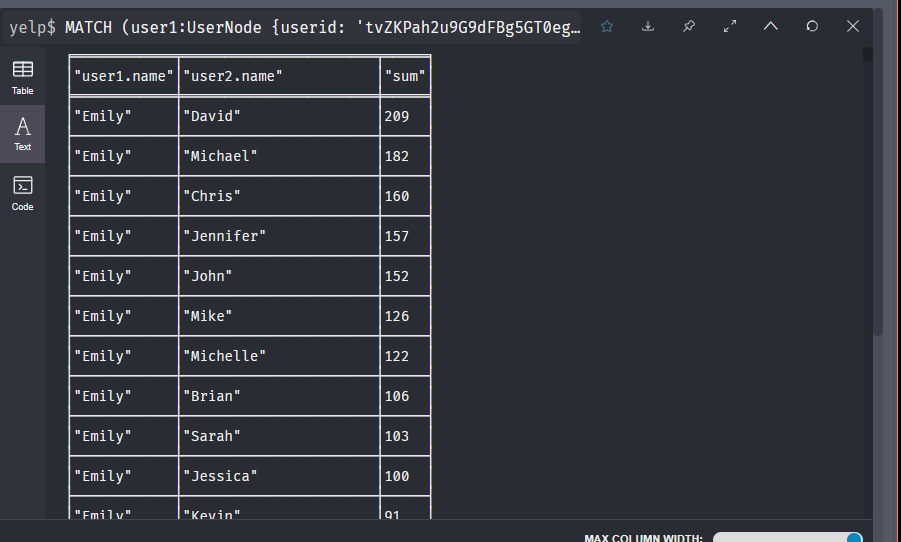
with user1, COLLECT(distinct b) as user1\_businesses

match (user2:UserNode)-[:Review]->(:ReviewNode)-[:Reviewed]->(b:BusinessNode)

where not (user1)-[:HasFriend]->(user2) and not (user2)-[:HasFriend]->(user1) and b in user1\_businesses

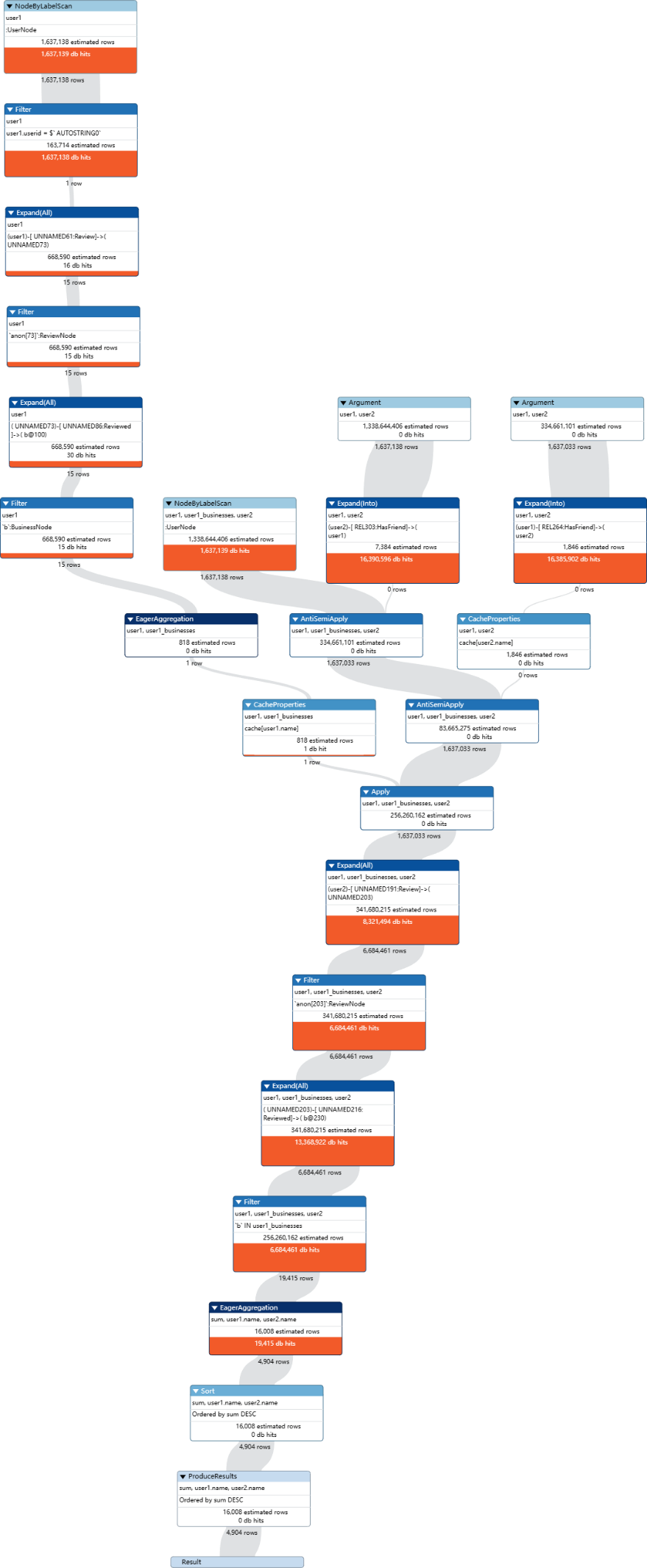
return user1.name, user2.name, count(b) as sum

order by sum desc



**（没有优化）Started streaming 4904 records after 140 ms and completed after 141 ms, displaying first 1000 rows.**

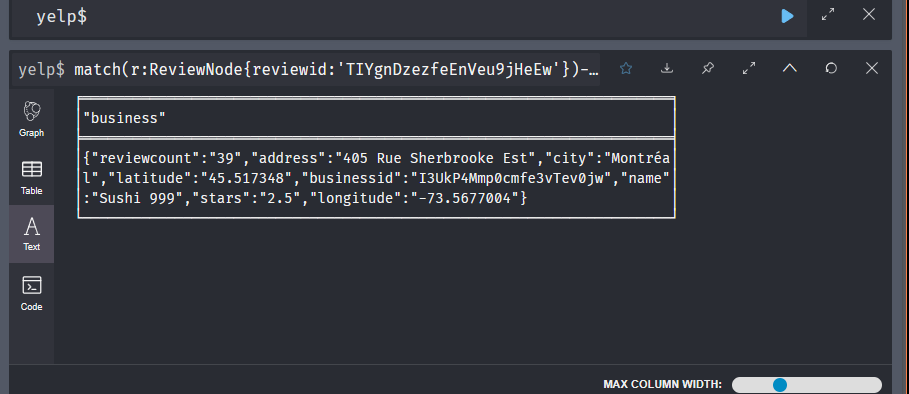
**对应的查询图：**



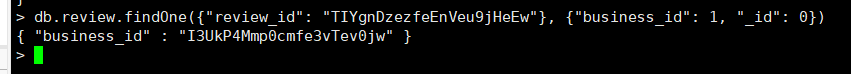
18. 分别使用Neo4j和MongoDB查询review\_id为TIYgnDzezfeEnVeu9jHeEw对 应的business信息，比较两者查询时间，指出Neo4j和MongoDB主要的适用场 景。

match(r:ReviewNode{r.reviewid:'TIYgnDzezfeEnVeu9jHeEw'})-[:Reviewed]->(business:BusinessNode) return business

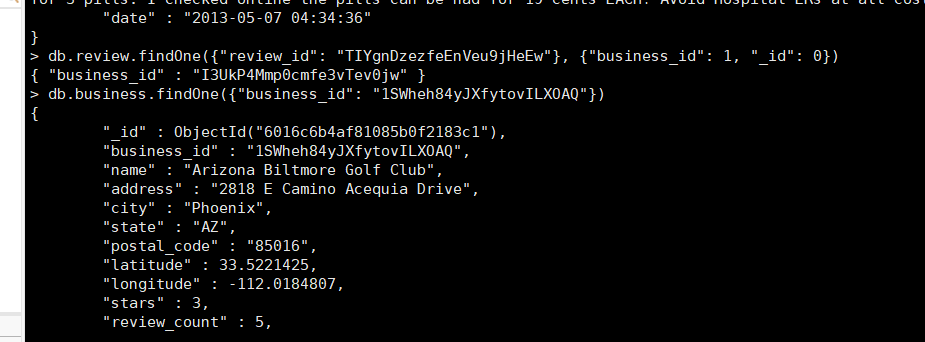
用时：Started streaming 1 records in less than 1 ms and completed after 14250 ms.



db.review.findOne({"review\_id": "TIYgnDzezfeEnVeu9jHeEw"}, {"business\_id": 1, "\_id": 0})



db.business.findOne({"business\_id": "1SWheh84yJXfytovILXOAQ"})



db.business.find({"business\_id": "1SWheh84yJXfytovILXOAQ"}).explain("executionStats")

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "test.business",

"indexFilterSet" : false,

"parsedQuery" : {

"business\_id" : {

"$eq" : "1SWheh84yJXfytovILXOAQ"

}

},

"winningPlan" : {

"stage" : "EOF"

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 0,

"executionTimeMillis" : 11,

"totalKeysExamined" : 0,

"totalDocsExamined" : 0,

"executionStages" : {

"stage" : "EOF",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1

}

},

"serverInfo" : {

"host" : "vector",

"port" : 27017,

"version" : "4.4.25",

"gitVersion" : "3e18c4c56048ddf22a6872edc111b542521ad1d5"

},

"ok" : 1

}