Yelp Dataset Analytics – Neo4j

MONGODB TO NEO4J GRAPH

SUBMITTED BY,
ANANNYA DAS
SRAVANI KOTHI
SHREYA PATIL
SHILPA VERMA

- **PURPOSE 1**

Considering the huge amount of dataset that yelp has, we decided on chopping all the json based on certain features. The procedure of the chopping is as mentioned below:

1. **Business.json**: Here, we have eliminated all those states that are not NV ie. we have kept the data of business that are located in the state of NV.

   ```
   db.business.remove( { state: {$ne:"NV"}  }
   ```

2. **User.json**: Here, we only take data of users whose review_count is greater than 200.

   ```
   db.user.remove({review_count: {$lt: 200}})
   ```

3. **Review.json:** Here, we only take the data of the reviews that are posted on dates after 2017-11-30.

   ```
   db.review.remove({date:{$lt:"2017-11-30"}})
   ```

4. **Tip.json**: Here, we only take the data of those tips that are posted on dates after 2017-10-31.

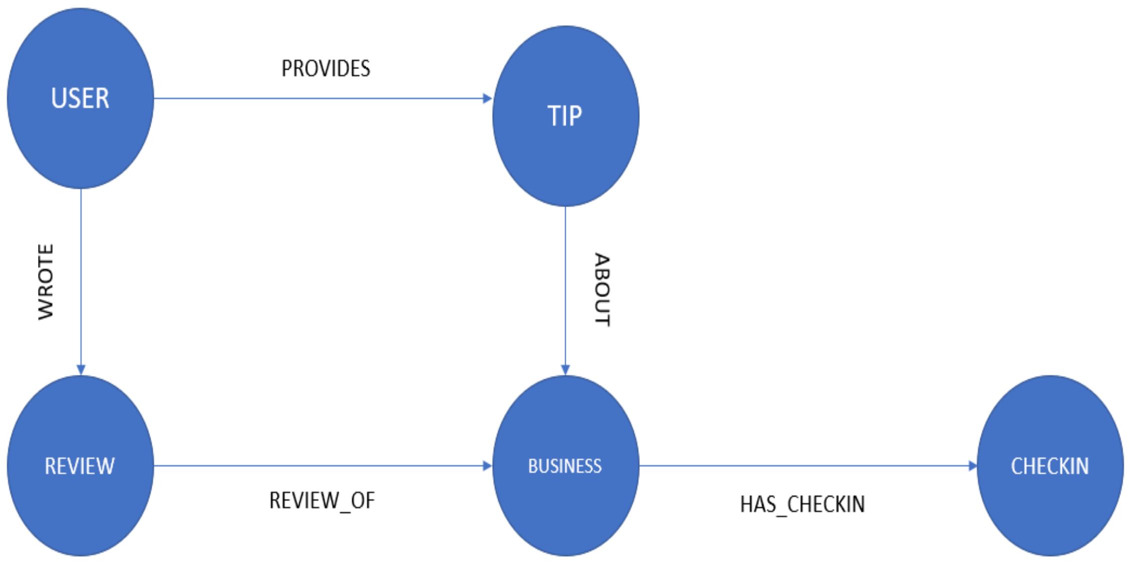
   ```
   db.tip.remove({date:{$lt:"2017-10-31"}})
   ```

5. **Checkin.json:** Here, we only consider the data of checkins that are not null on friday, saturday and sunday.

   ```
   db.checkin.remove({$and:[{'time.Friday':{$ne:null}},{'time.Saturday':{$ne:null}},{'time.Sunday':{$ne:null}}]})
   ```

The purpose behind this pattern of chopping is to maintain useful data so as to answer most of the queries that can raised based on the graph model and for easy and convenient traversal.

PROPOSED GRAPH DATA MODEL



I. Summary of Graph Model:

The graph model consists of 5 nodes and 5 edges/relationships. Their descriptions are as below:

Nodes:

1. User

Properties:

_id, user_id, name, review_count, yelping_since, friends, ,funny,cool,fans,elite,average_stars

2. Review

Properties:

_id,review_id,user_id,business_id,stars,date,text

3. Business

Properties:

_id,business_id,name,address,city,state,postal_code,latitude,longitude,stars,review_count,categories,hours

4. **Tip**

Properties:

_id,text,date,likes,business_id,user_id

5. **Checkin**

Properties: _id,time,business_id

Relationships:

1. PROVIDES
2. WROTE
3. REVIEW_OF
4. HAS_CHECKIN
5. ABOUT

Comparison of graph model with mini-project 4:

In mini-project 4 we have created category as a different node but it is not required since we can query category as a property of business node. So, we removed category as a node. Also, we created tip as different node and identified relationship User to Tip and Tip to Business.

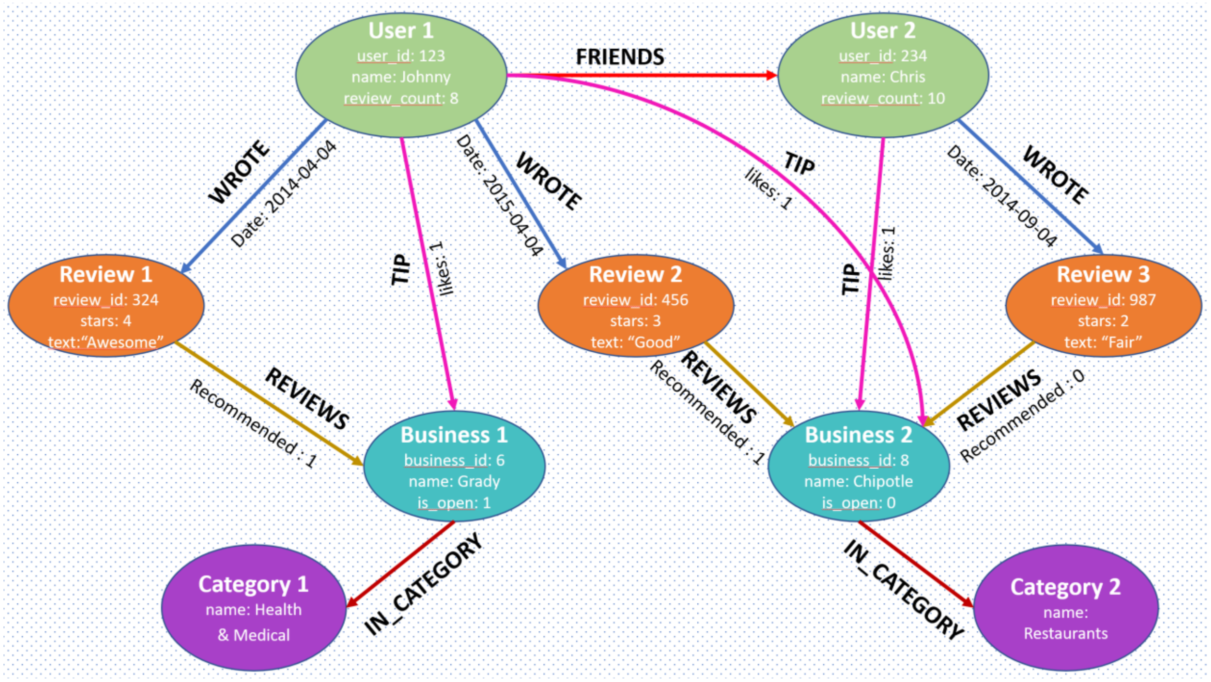


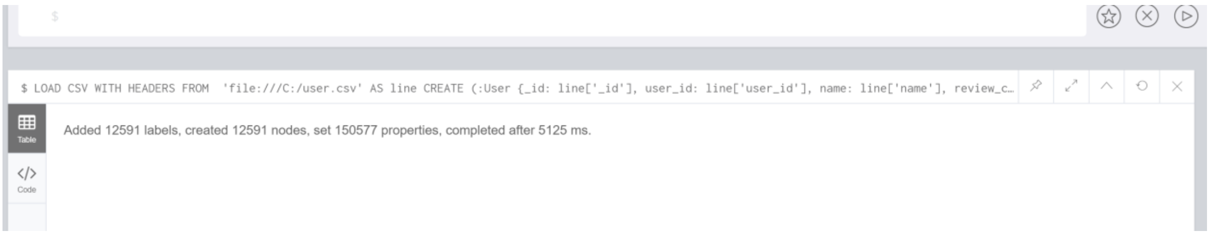
Figure 1 Previous Data Model

II. Node Creation:

Justification: To import data from a CSV file into Neo4j, we are using use LOAD CSV to get the data into your query. Then we write it to our database using the clauses of Cypher.

1. **Creating node User by loading data from user.csv**

```
LOAD CSV WITH HEADERS FROM
'file:///C:/user.csv' AS line
CREATE (:User {_id: line['_id'], user_id: line['user_id'], name: line['name'],
review_count: line['review_count'], yelping_since: line['yelping_since'], friends:
line['friends'], useful: line['useful'], funny: line['funny'], cool: line['cool'], fans:
line['fans'], elite: line['elite'], average_stars: line['average_stars']})
```



```
$ LOAD CSV WITH HEADERS FROM  'file:///C:/user.csv' AS line CREATE (:User {_id: line['_id'], user_id: line['user_id'], name: line['name'], review_c…
```

```
Added 12591 labels, created 12591 nodes, set 150577 properties, completed after 5125 ms.
```

2. **Creating node Checkin by loading data from checkin.csv**

```
LOAD CSV WITH HEADERS FROM
'file:///checkin.csv' AS line
CREATE (:Checkin { _id: line['_id'], time: line['time'], business_id:
line['business_id']})
```

3. **Creating node Review by loading data from review.csv**

```
LOAD CSV WITH HEADERS FROM
'file:///C:/review.csv' AS line
CREATE (:Review {_id: line['_id'],review_id: line['review_id'], user_id:
line['user_id'],business_id: line['business_id'],stars: line['stars'], date:
line['date'],   text: line['text']})
```

4. **Creating node Business by loading data from business.csv**

```
LOAD CSV WITH HEADERS FROM
'file:///C:/business.csv' AS line
CREATE (:Business {_id: line['id'], business_id: line['business_id'], name:
line['name'],    address: line['address'], city: line['city'], state:
line['state'], postal_code:       line['postal_code'], latitude: line['latitude'], longitude:
line['longitude'], stars:            line['stars'], review_count: line['review_count'],
categories: line['categories'], hours: line['hours']})
```

5. **Creating node Tip by loading data from tip.csv**

```
LOAD CSV WITH HEADERS FROM
'file:///tip.csv' AS line
CREATE (:Tip { _id: line['_id'], text: line['text'],date: line['date'], likes: line['likes'],
        business_id: line['business_id'], user_id: line['user_id']})
```

Relationship creation:

Justification: The statement for creating a relationship consists of CREATE, followed by the details of the relationship that we are creating.

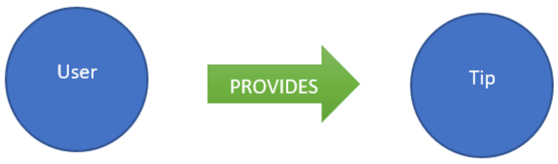
Let's create a relationship between some of the nodes that we created above.

The MATCH will lookup up all nodes carrying the label and see if the property matches.

1. **User provides Tip**

```
MATCH (n:Tip), (m:User) where n.user_id = m.user_id create (n)<-[r:PROVIDES]-(m)
RETURN n,m,r
```

Explanation of above code:



First, we use a MATCH statement to find the two nodes that we want to create the relationship between.

We use the user_id property that we'd previously assigned to each node to establish relationship between nodes.

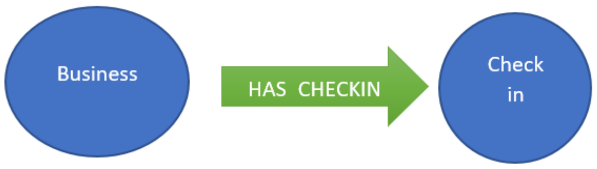
Then there's the actual CREATE statement. This is what creates the relationship. In this case, it references the two nodes by the variable name (i.e. n and m) that we gave them in the first line. The relationship is established by using an ASCII-code pattern, with an arrow indicating the direction of the relationship: (m)-[r:PROVIDES]->(n).

We give the relationship a variable name of r and give the relationship a type of PROVIDES.

2. **Business has Checkin**

```
Match (a:Checkin),(b:Business) where a.business_id=b.business_id create (a)<-
[has:HAS_CHECKIN]-(b) return a,b, has
```

Explanation of above code:



First, we use a MATCH statement to find the two nodes that we want to create the relationship between.

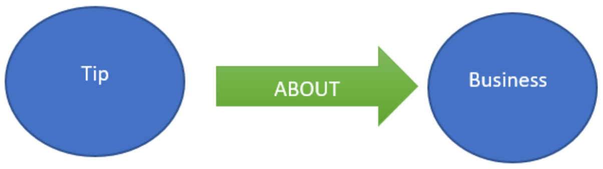
We use the business_id property that we'd previously assigned to each node to establish relationship between them.

Then there's the actual CREATE statement. This is what creates the relationship. In this case, it references the two nodes by the variable name (i.e. a and b) that we gave them in the first line. The relationship is established by using an ASCII-code pattern, with an arrow indicating the direction of the relationship: (b)-[r:HAS_CHECKIN]->(a).

We give the relationship a variable name of r and give the relationship a type of PROVIDES.

3. **Tip about Business**
   MATCH (n:Tip), (m:Business) where n.business_id = m.business_id create (m)<-[r:ABOUT]-(n) RETURN n,m,r

   Explanation of above code:



First, we use a MATCH statement to find the two nodes that we want to create the relationship between.

We use the business_id property that we'd previously assigned to each node to establish relationship between them..

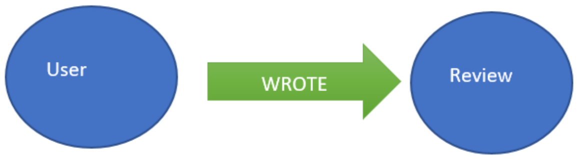
Then there's the actual CREATE statement. This is what creates the relationship. In this case, it references the two nodes by the variable name (i.e. n and m) that we gave them in the first line. The relationship is established by using an ASCII-code pattern, with an arrow indicating the direction of the relationship: (n)-[r:ABOUT]->(m).

We give the relationship a variable name of r and give the relationship a type of ABOUT.

4. **User wrote Review**

   Match (a:User),(b:Review) where a.user_id=b.user_id create (a)-[has:WROTE]->(b) return a,b, has

   Explanation of above code:



First, we use a MATCH statement to find the two nodes that we want to create the relationship between.

We use the user_id property that we'd previously assigned to each node to establish relationship between them.

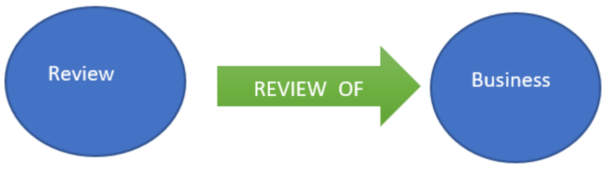
Then there's the actual CREATE statement. This is what creates the relationship. In this case, it references the two nodes by the variable name (i.e. a and b) that we gave them in the first line. The relationship is established by using an ASCII-code pattern, with an arrow indicating the direction of the relationship: (a)-[r:WROTE]->(b).

We give the relationship a variable name of r and give the relationship a type of WROTE.

5. **Review is the review of the Business**

   Match (a:Review),(b:Business) where a.business_id=b.business_id create (a)-[has:REVIEW_OF]->(b) return a,b, has

   Explanation of above code:



First, we use a MATCH statement to find the two nodes that we want to create the relationship between.

We use the business_id property that we'd previously assigned to each node to establish relationship between them.

Then there's the actual CREATE statement. This is what creates the relationship. In this case, it references the two nodes by the variable name (i.e. a and b) that we gave them in the first line. The relationship is established by using an ASCII-code pattern, with an arrow indicating the direction of the relationship: (a)-[r:REVIEW_OF]->(b).

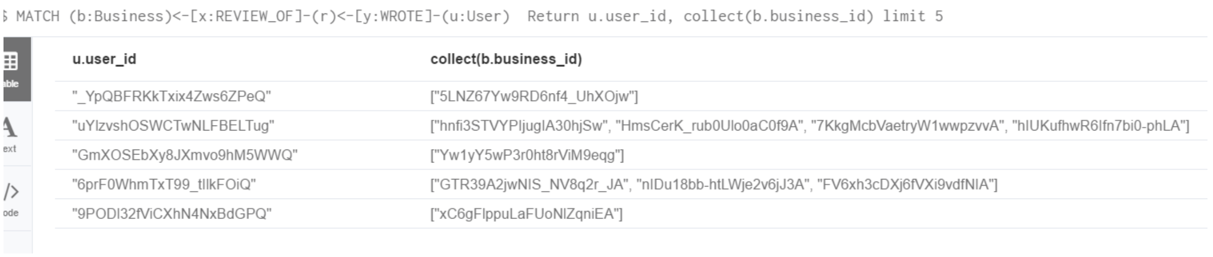
We give the relationship a variable name of r and give the relationship a type of REVIEW_OF.

- **<u>Purpose-2:</u>**

1. **Identify the businesses that a reviewer has reviewed (limit to 5 reviewers)**

MATCH (b:Business)<-[x:REVIEW_OF]-(r)<-[y:WROTE]-(u:User)

Return u.user_id, collect(b.business_id) limit 5



```
$ MATCH (b:Business)<-[x:REVIEW_OF]-(r)<-[y:WROTE]-(u:User)  Return u.user_id, collect(b.business_id) limit 5
```

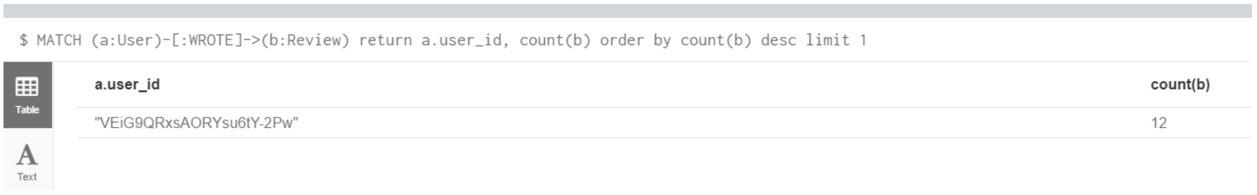
u.user_id	collect(b.business_id)
"_YpQBFRKkTxix4Zws6ZPeQ"	["5LNZ67Yw9RD6nf4_UhXOjw"]
"uYlzvshOSWCTwNLFBELTug"	["hnfi3STVYPljuglA30hjSw", "HmsCerK_rub0Ulo0aC0f9A", "7KkgMcbVaetryW1wwpzvvA", "hIUKufhwR6lfn7bi0-phLA"]
"GmXOSEbXy8JXmvo9hM5WWQ"	["Yw1yY5wP3r0ht8rViM9eqg"]
"6prF0WhmTxT99_tllkFOiQ"	["GTR39A2jwNIS_NV8q2r_JA", "nIDu18bb-htLWje2v6jJ3A", "FV6xh3cDXj6fVXi9vdfNIA"]
"9PODI32fViCXhN4NxBdGPQ"	["xC6gFlppuLaFUoNlZqniEA"]

Explanation of above code:

We traverse the graph starting from User node (denoted by u), we find all the businesses the particular User u has reviewed. We then return the business ids of all businesses (denoted by b) of which the User u has posted a review about. We use the limit parameter to limit our result set to 5 results.

2. **Identify the reviewer with the most number of reviews**

MATCH (a:User)-[:WROTE]- >(b:Review)
return a.user_id, count(b)
order by count(b) desc limit 1



```
$ MATCH (a:User)-[:WROTE]->(b:Review) return a.user_id, count(b) order by count(b) desc limit 1
```

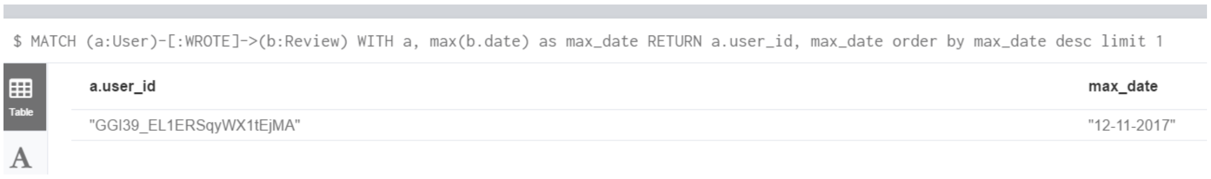
a.user_id	count(b)
"VEiG9QRxsAORYsu6tY-2Pw"	12

Explanation of above code:

Here we traverse the graph starting with node User (denoted by a) and based on the relationship (denoted by :WROTE) find all reviews of the particular User a. In the return statement we return the user_id property of User a alongwith the total count of reviews written by User a. We then use the order by function to sort the total count of reviews in descending order to find the user with the most number of reviews.

3. Identify the reviewer with the newest review

MATCH (a:User)-[:WROTE]->(b:Review)
WITH a, max(b.date) as max_date
RETURN a.user_id, max_date
order by max_date desc limit 1



```
$ MATCH (a:User)-[:WROTE]->(b:Review) WITH a, max(b.date) as max_date RETURN a.user_id, max_date order by max_date desc limit 1
```

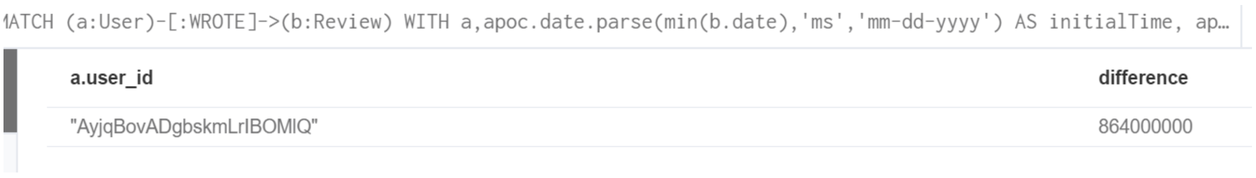
a.user_id	max_date
"GGI39_EL1ERSqyWX1tEjMA"	"12-11-2017"

Explanation of above code:

Here we traverse the graph starting with node User (denoted by a) and based on the relationship (denoted by :WROTE) find all reviews of the particular User a. In the return statement we return the user_id property of User a alongwith the most recent review date, using function max() as max_date of the review posted by User a. We then use the order by function to sort the max_date of reviews in descending order to find the user with the most recent review.

4. Identify the reviewer who has been reviewing for the longest period of time

MATCH (a:User)-[:WROTE]->(b:Review)
WITH a,apoc.date.parse(min(b.date),'ms','mm-dd-yyyy') AS initialTime, apoc.date.parse(max(b.date),'ms','mm-dd-yyyy') AS finalTime
RETURN a.user_id,finalTime - initialTime as difference
order by difference desc limit 1



```
ATCH (a:User)-[:WROTE]->(b:Review) WITH a,apoc.date.parse(min(b.date),'ms','mm-dd-yyyy') AS initialTime, ap…
```

a.user_id	difference
"AyjqBovADgbskmLrIBOMlQ"	864000000

Explanation of above code:

Here we traverse the graph starting with node User (denoted by a) and based on the relationship (denoted by :WROTE) find all reviews of the particular User a.

We import the apoc procedure library to use the inbuilt function apoc.date.parse() to find the newest and the oldest review date of the particular user(denoted by a).

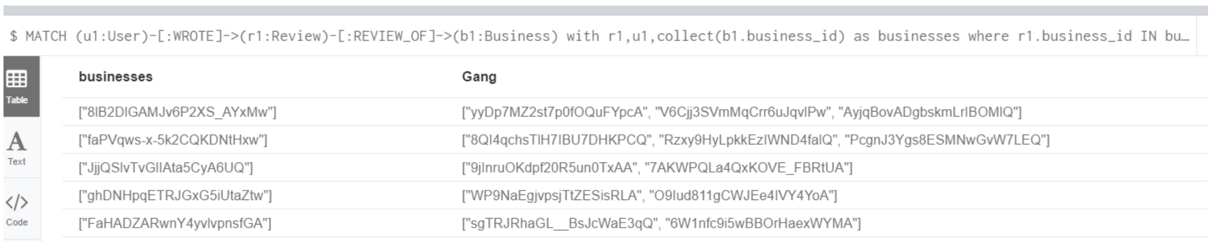
We parse the oldest review date using apoc.date.parse(min(b.date),'ms','mm-dd-yyyy') AS initialTime, and the newest review date using apoc.date.parse(max(b.date),'ms','mm-dd-yyyy') AS finalTime.

In the return statement we return the user_id property of User a alongwith the date difference between finalTime and initialTime as difference of the review time span posted by User a.

We then use the order by function to sort the date difference (denoted by difference) of review post dates of User a in descending order to find the user with the who has been reviewing for the longest period of time.

5. Identify the reviewers who reviewed a common set of businesses.

```
MATCH (u1:User)-[:WROTE]->(r1:Review)-[:REVIEW_OF]->(b1:Business)
with r1,u1,collect(b1.business_id) as businesses
where r1.business_id IN businesses
WITH businesses,collect(u1.user_id) as Gang
return businesses, Gang
```



```
$ MATCH (u1:User)-[:WROTE]->(r1:Review)-[:REVIEW_OF]->(b1:Business) with r1,u1,collect(b1.business_id) as businesses where r1.business_id IN bu…
```

businesses	Gang
["8lB2DIGAMJv6P2XS_AYxMw"]	["yyDp7MZ2st7p0fOQuFYpcA", "V6Cjj3SVmMqCrr6uJqvlPw", "AyjqBovADgbskmLrlBOMIQ"]
["faPVqws-x-5k2CQKDNtHxw"]	["8QI4qchsTIH7IBU7DHKPCQ", "Rzxy9HyLpkkEzIWND4falQ", "PcgnJ3Ygs8ESMNwGvW7LEQ"]
["JjjQSlvTvGIIAta5CyA6UQ"]	["9jlnruOKdpf20R5un0TxAA", "7AKWPQLa4QxKOVE_FBRtUA"]
["ghDNHpqETRJGxG5iUtaZtw"]	["WP9NaEgjvpsjTtZESisRLA", "O9lud811gCWJEe4lVY4YoA"]
["FaHADZARwnY4yvlvpnsfGA"]	["sgTRJRhaGL__BsJcWaE3qQ", "6W1nfc9i5wBBOrHaexWYMA"]

Explanation of above code:

Here we traverse the graph starting with node User (denoted by u1) and to Business denoted by b1. We are collecting all businesses reviewed by a user. And if the same business is reviewed by another user we are making set of those businesses and users. And returning set of businesses as businesses and set of users as Gang.