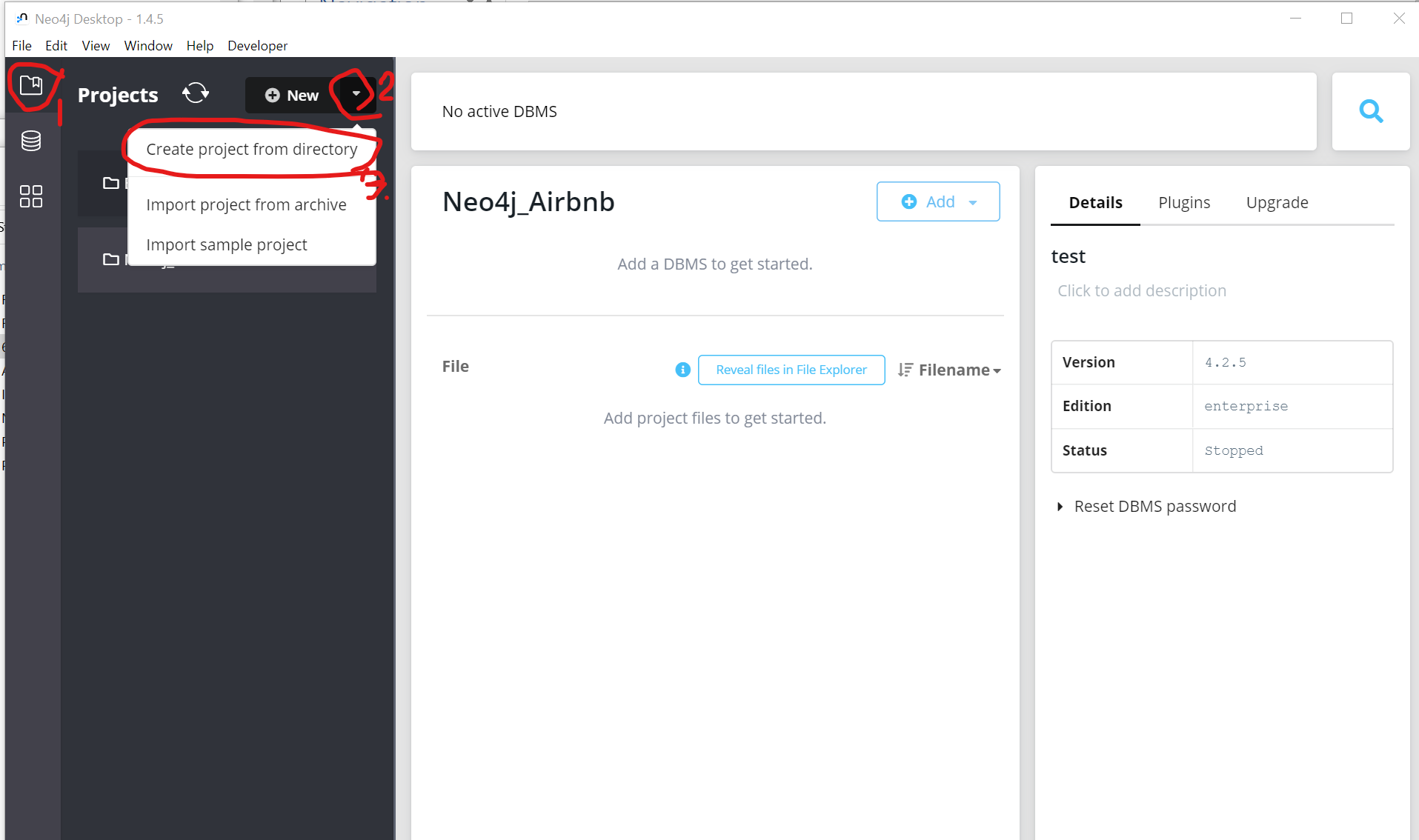
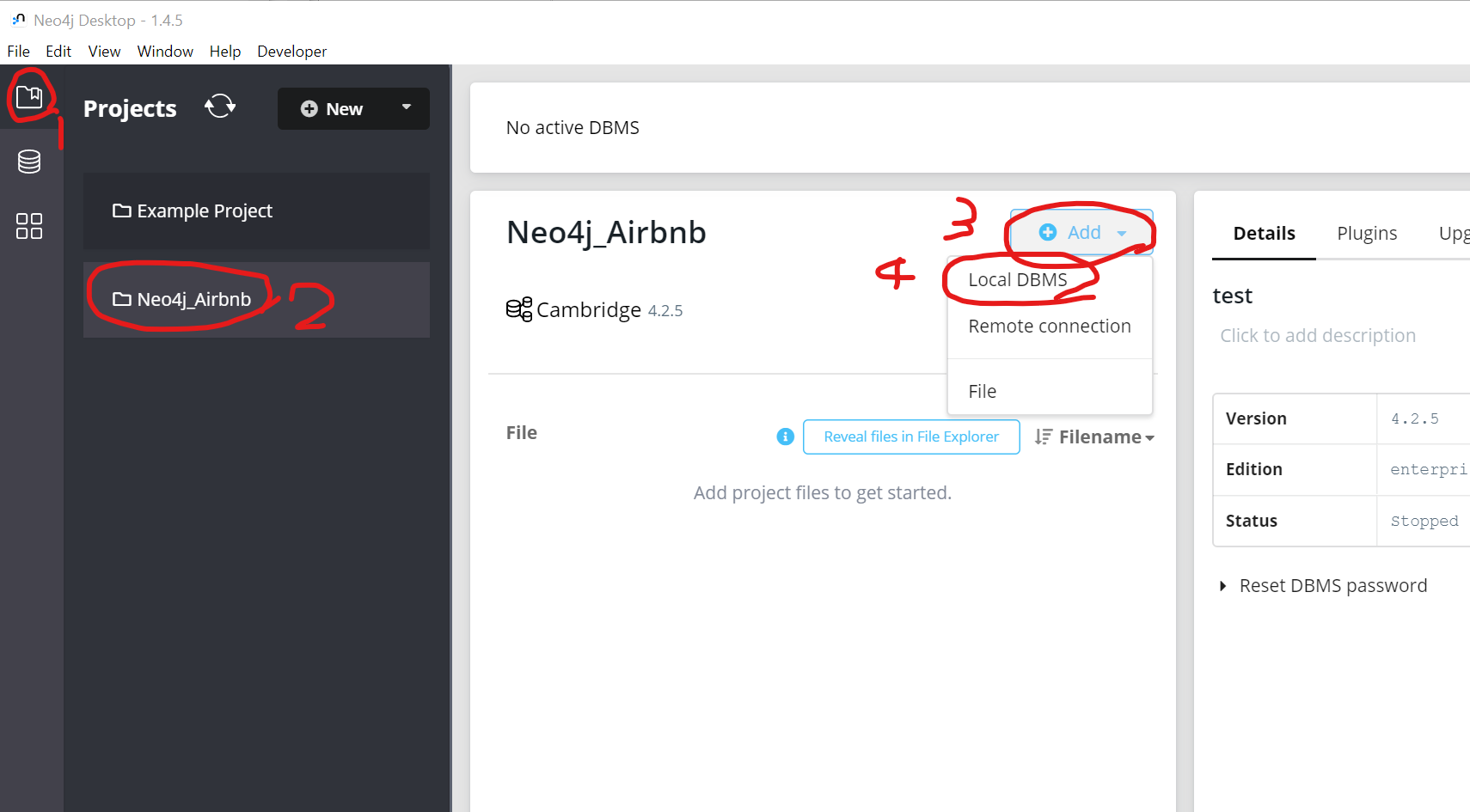
# 1. Installation, configure, and connect

(1) Install and start Neo4j desktop

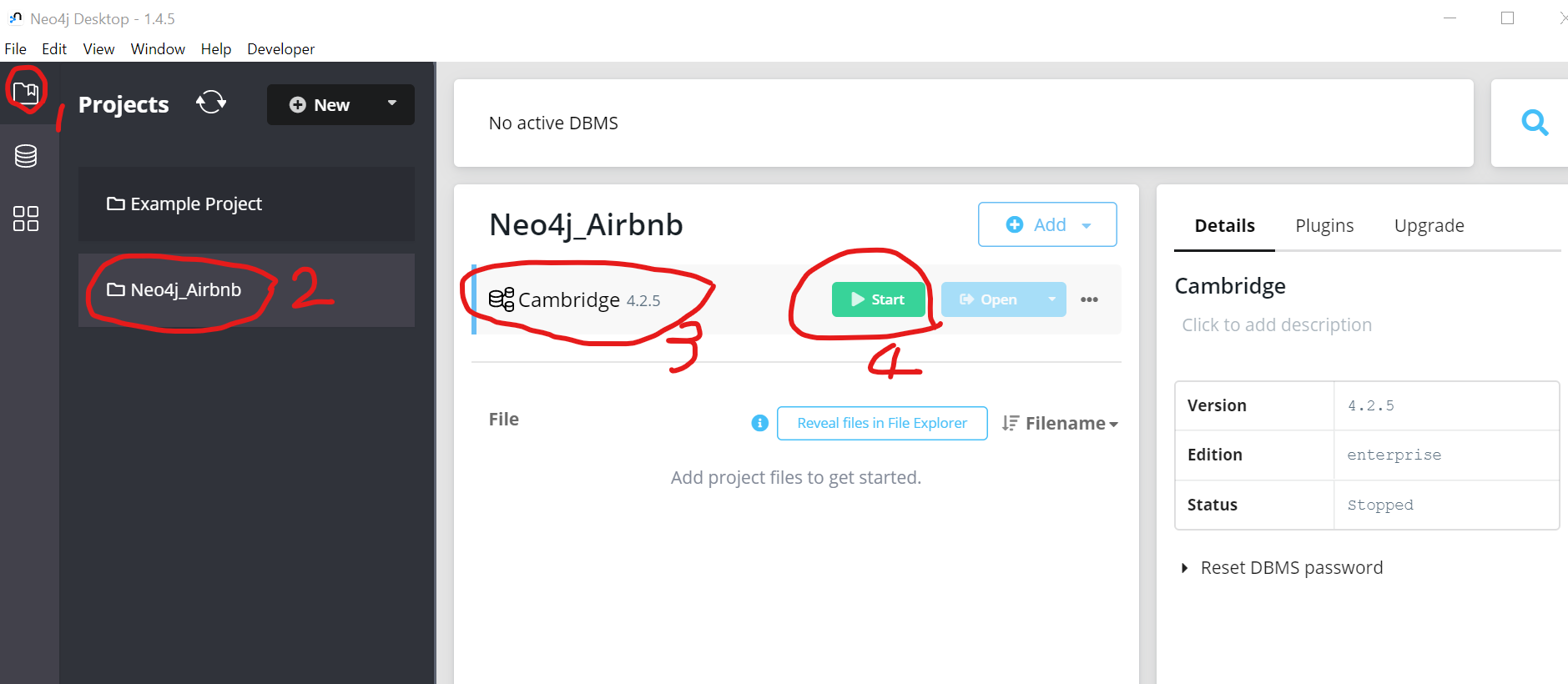
(2) Create a folder “Neo4j\_Airbnb” as a new project

****

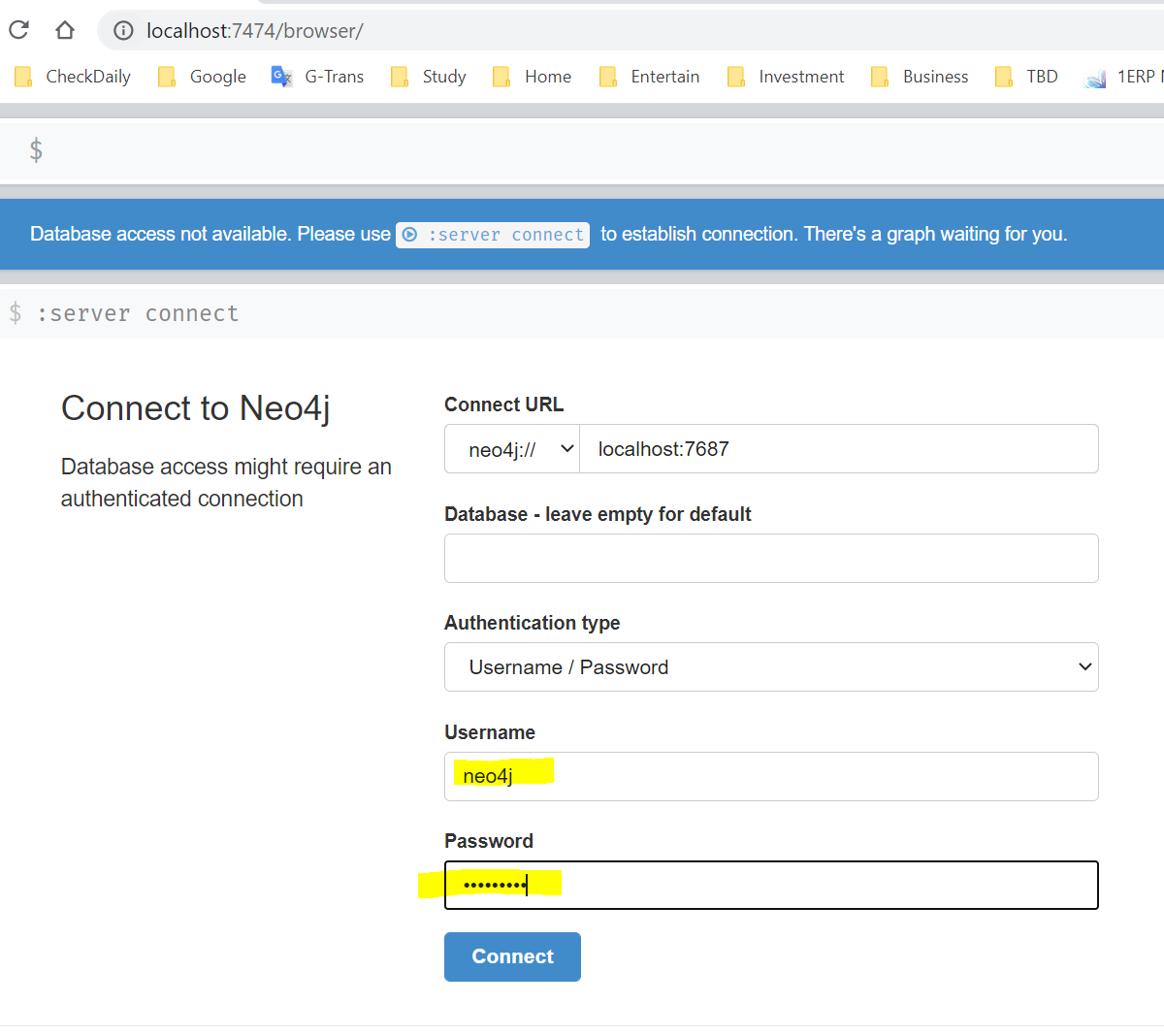
(3) Create a database “Cambridge” under project Neo4j\_Airbnb (password “Cambridge”)



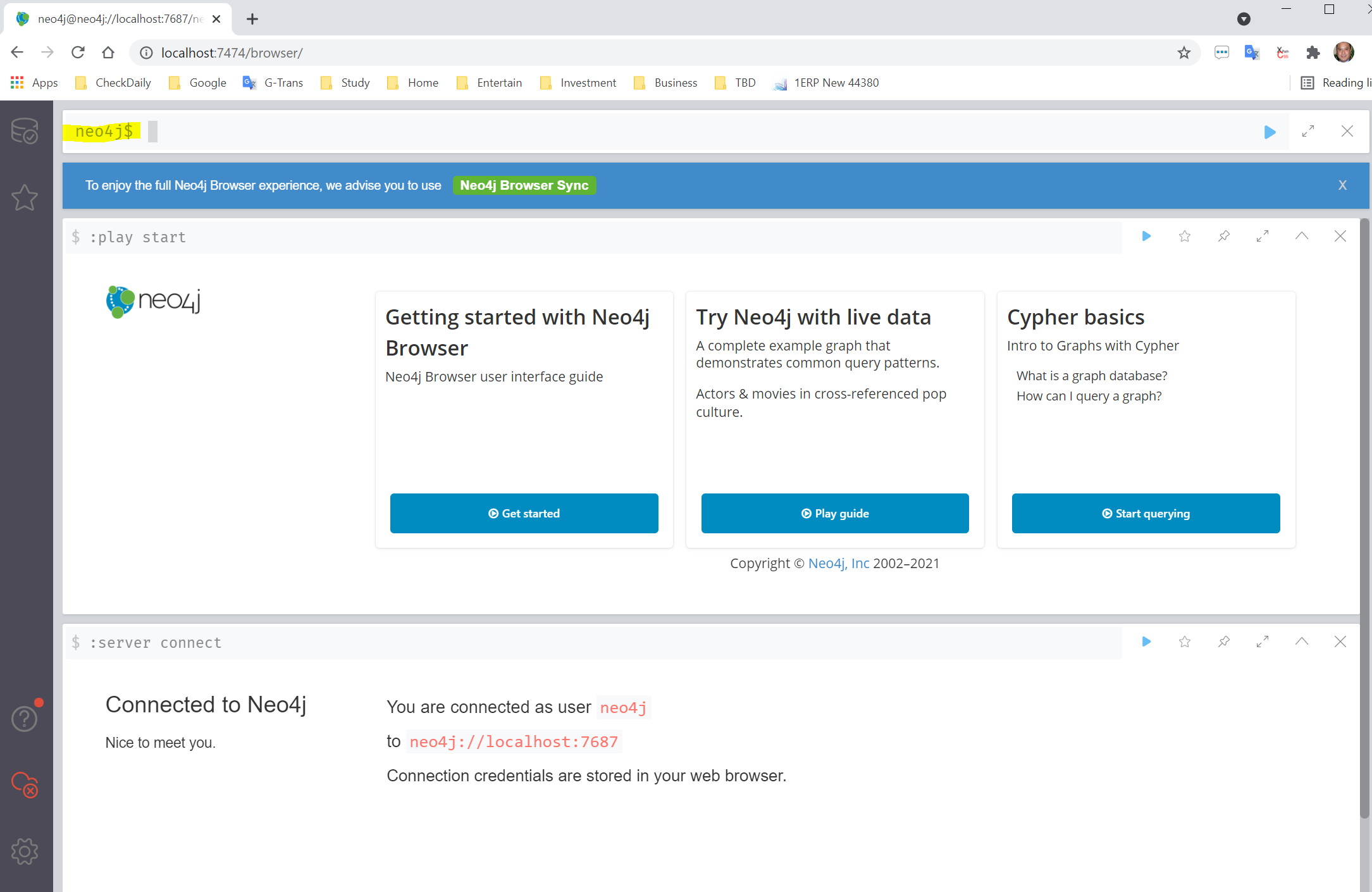
(4) Start the database “Cambridge”



(5) Start a browser, and enter <http://localhost:7474/browser/>, use “neo4j” as user, “Cambridge” as password to login.



(6) Now you can get this interface in a browser, where you can enter command to manipulate the database.



# 2. Data Creation in Cypher

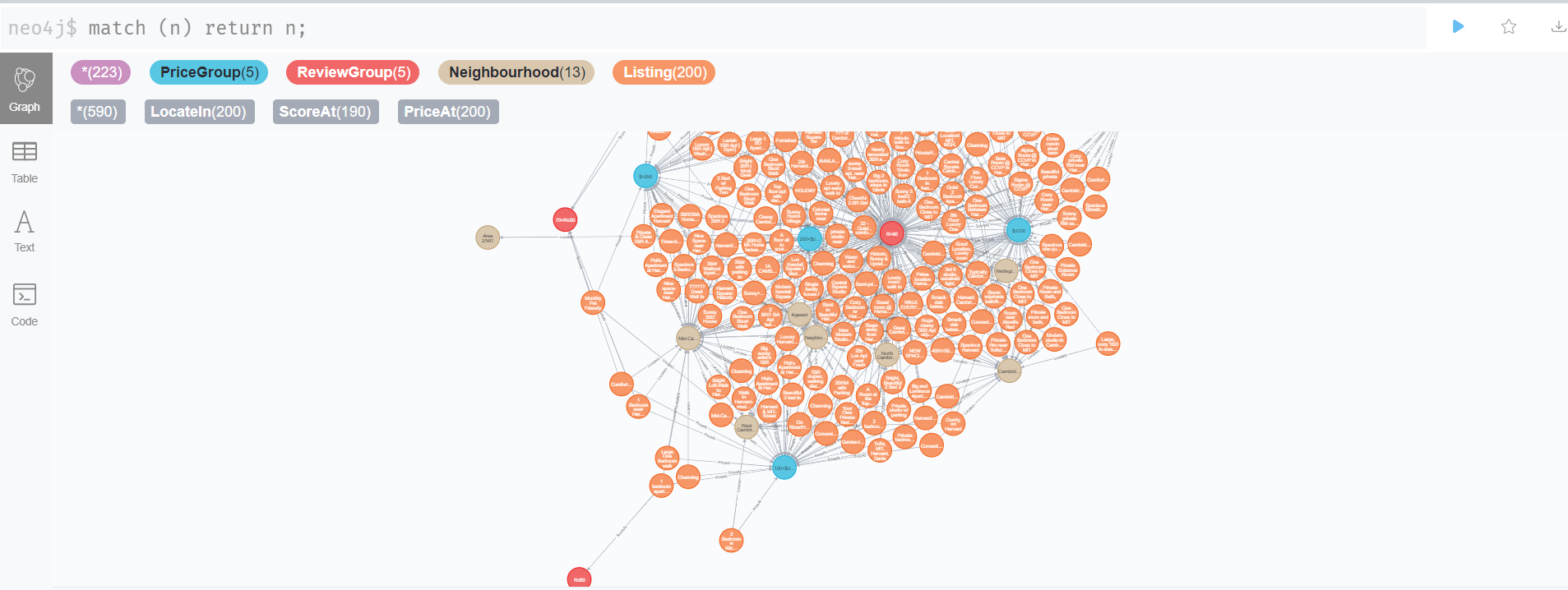
## (1) Full data set: Neo4j.Create.Fullset.txt

Total there are 817 listings, 5 price group, 5 review score group, and 13 neighborhood, and 2500 edges.

During loading process, I have spent like 4 hours to build above nodes and edges, but still cannot finish, so I decide to use a subset of the data.

## (2) Subset: Neo4j.Create.Subset.txt

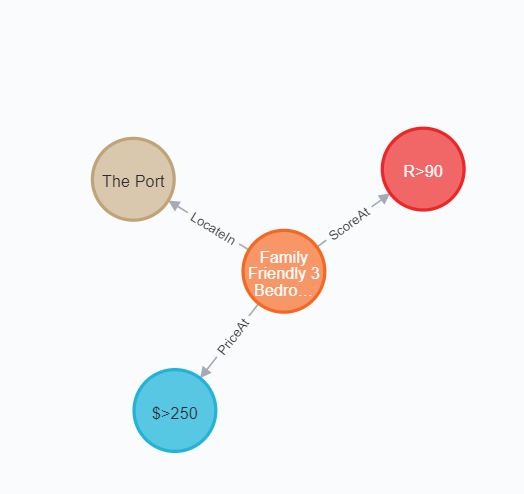
I choose 200 out of the 817 listings, which end up 600 edges, and I can load them within 1.5 hours.



# 3. Cypher Query

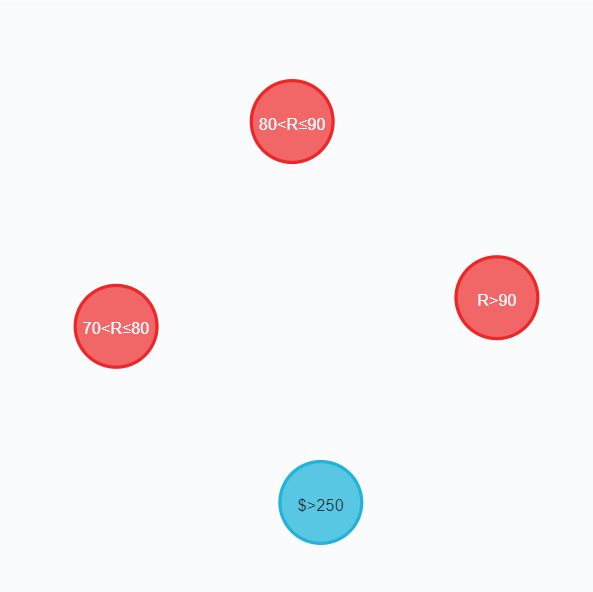
## (1) Query a listing with its edges to price group, review group, and neighborhood

match(l:Listing {id:4455298}),(p:PriceGroup),(s:ReviewGroup),(n:Neighbourhood) return l,p,s,n;



## (2) Query how many review groups are corresponding to a price group

match (p:PriceGroup {name:'$>250'})<-[:PriceAt]-(l:Listing)-[:ScoreAt]->(s:ReviewGroup) return p,s;

## (3) How many review groups are corresponding to a neighbourhood

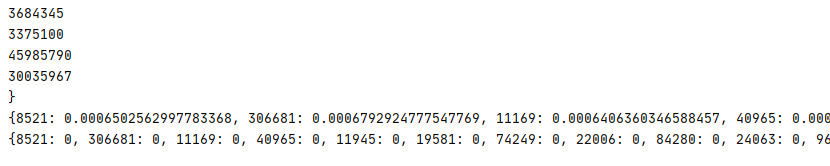
match (n:Neighbourhood {name:'Riverside'})<-[:LocateIn]-(l:Listing)-[:ScoreAt]->(s:ReviewGroup) return n,s;

## 

# 4.Algorithms

# we apply three graph algorithms to the graph model we construct; we use network to implement the graph algorithms, follow is the code and result





# 5. Analytics

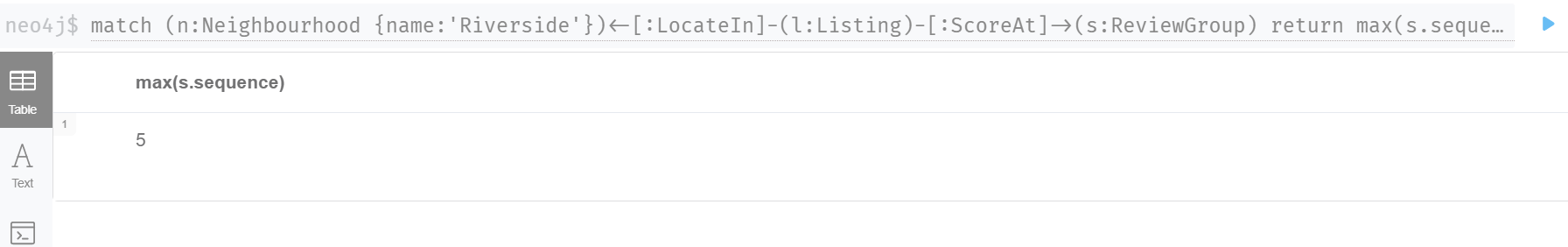
## (1) Giving a neighbourhood, get all its listings associated with max scored review group

### Method 1: Two-phase query

**(1) Get max review group (highest score, highest sequence)**

match (n:Neighbourhood {name:'Riverside'})<-[:LocateIn]-(l:Listing)-[:ScoreAt]->(s:ReviewGroup)

return max(s.sequence);

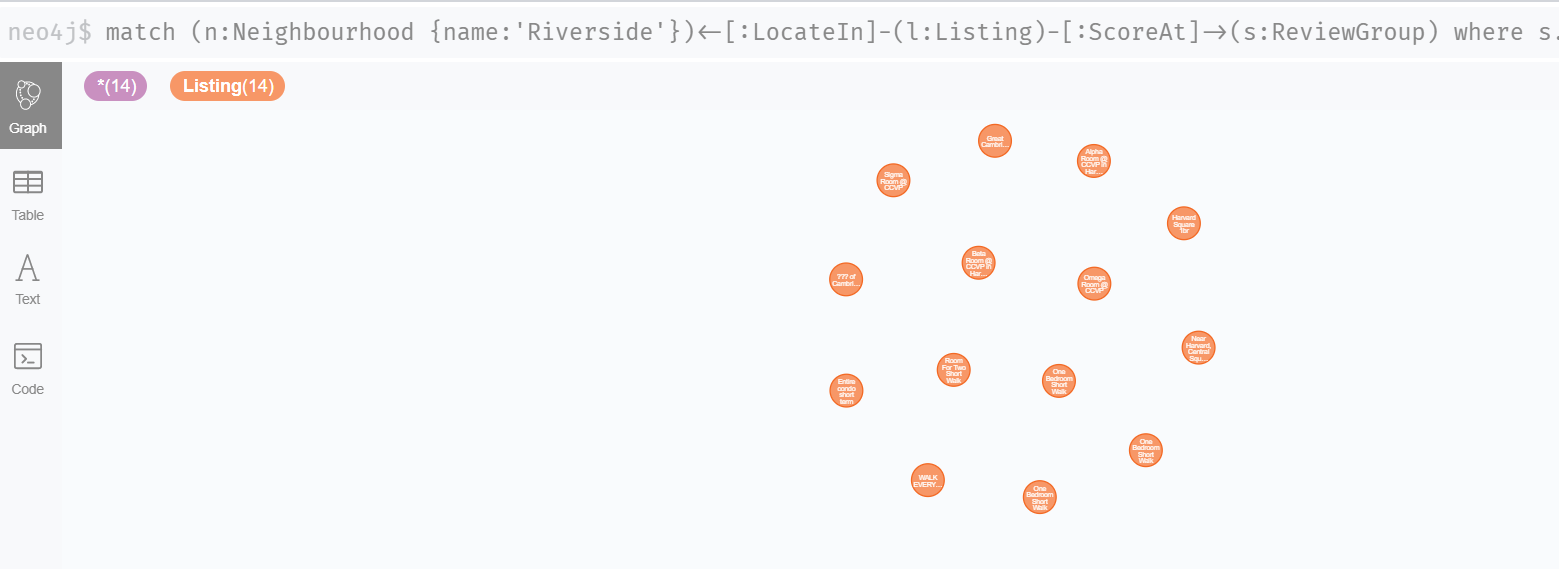


**(2) Use result from (1) to get all listings**

match (n:Neighbourhood {name:'Riverside'})<-[:LocateIn]-(l:Listing)-[:ScoreAt]->(s:ReviewGroup)

where s.sequence=5

return l;



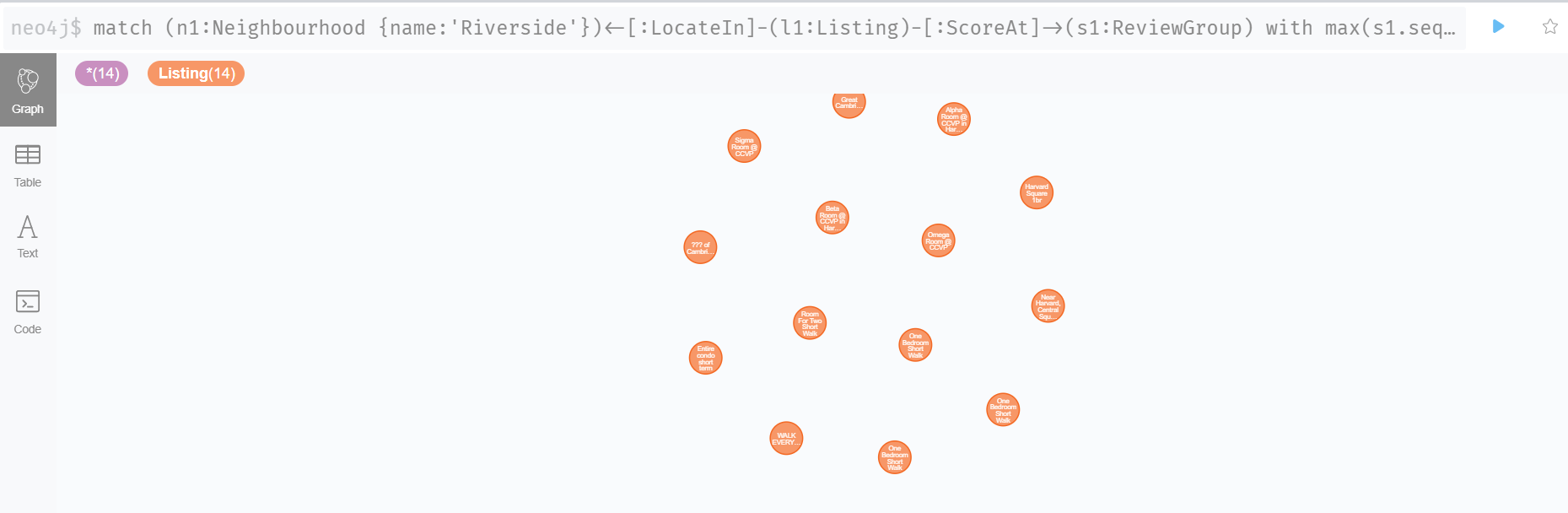
### Method 2: Merge above 2-step into one Cypher query

match (n1:Neighbourhood {name:'Riverside'})<-[:LocateIn]-(l1:Listing)-[:ScoreAt]->(s1:ReviewGroup)

with max(s1.sequence) as MaxSeq

match (n:Neighbourhood {name:'Riverside'})<-[:LocateIn]-(l:Listing)-[:ScoreAt]->(s:ReviewGroup)

where s.sequence=MaxSeq return l;



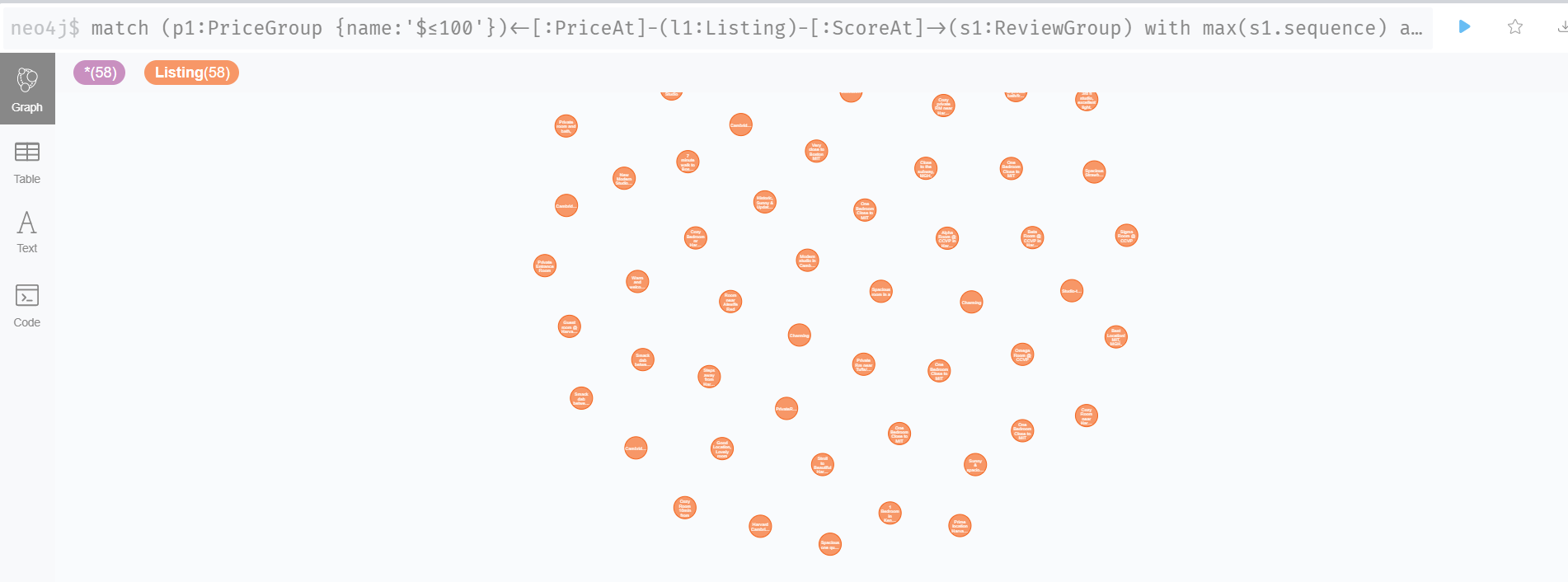
## (2) What are the listings associated with max review score group for the price group $<=100?

match (p1:PriceGroup {name:'$≤100'})<-[:PriceAt]-(l1:Listing)-[:ScoreAt]->(s1:ReviewGroup)

with max(s1.sequence) as MaxSeq

match (p:PriceGroup {name:'$≤100'})<-[:PriceAt]-(l:Listing)-[:ScoreAt]->(s:ReviewGroup)

where s.sequence=MaxSeq return l;



We can project edges (WithMaxReviewScore) between PriceGroup to Listings, or between Neighbourhood and Listings, this seems to be done via "match..... create...." but this is just performance improvement (save one hop), I will skip it for now.