**基于neo4j的Louvain算法的实现**

1. 打开Neo4j sandbox并新建一个新项目，然后Open with Browser
2. 导入数据集

CREATE

(nAlice:User {name: 'Alice', seed: 42}),

(nBridget:User {name: 'Bridget', seed: 42}),

(nCharles:User {name: 'Charles', seed: 42}),

(nDoug:User {name: 'Doug'}),

(nMark:User {name: 'Mark'}),

(nMichael:User {name: 'Michael'}),

(nAlice)-[:LINK {weight: 1}]->(nBridget),

(nAlice)-[:LINK {weight: 1}]->(nCharles),

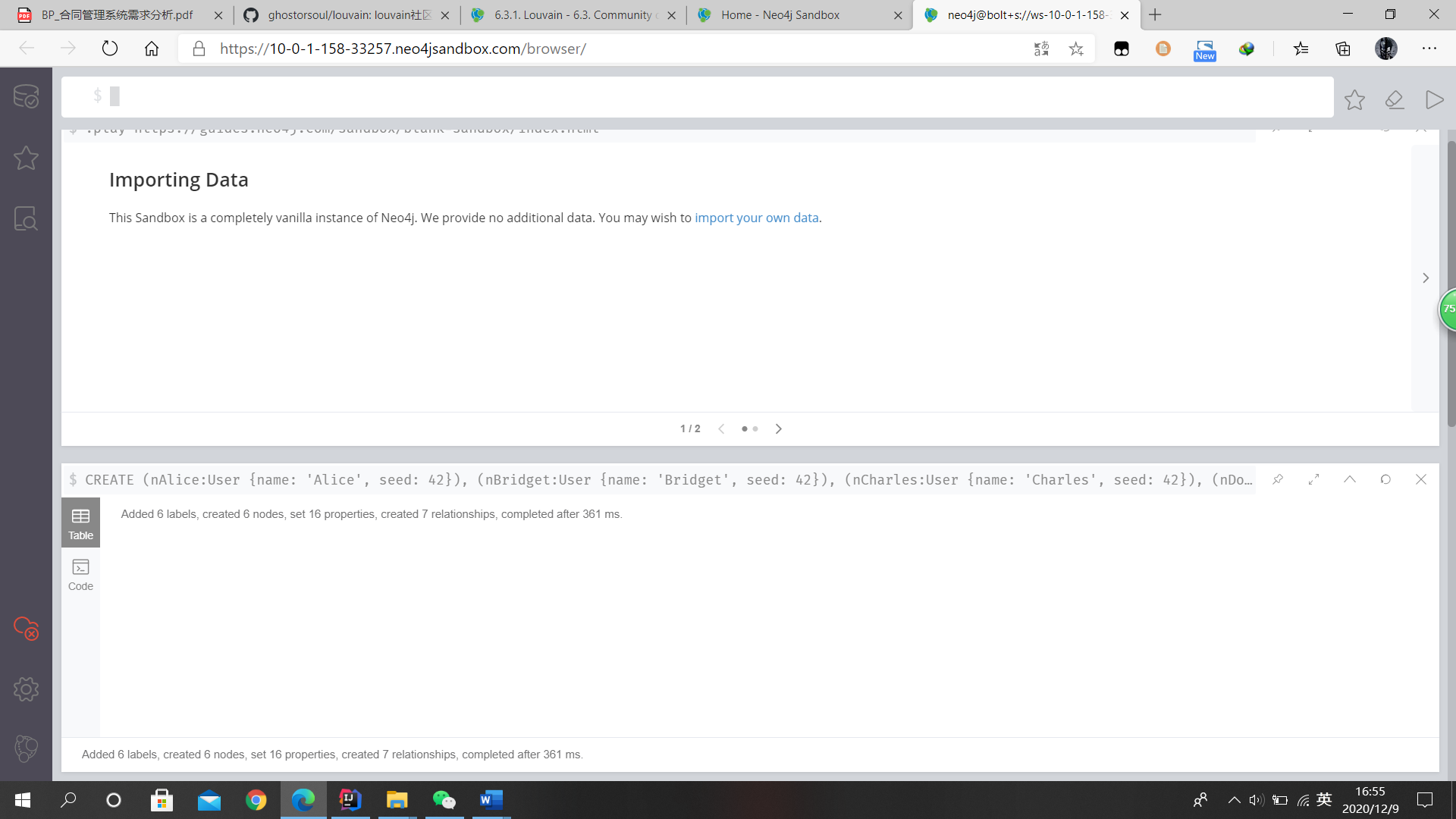
(nCharles)-[:LINK {weight: 1}]->(nBridget),

(nAlice)-[:LINK {weight: 5}]->(nDoug),

(nMark)-[:LINK {weight: 1}]->(nDoug),

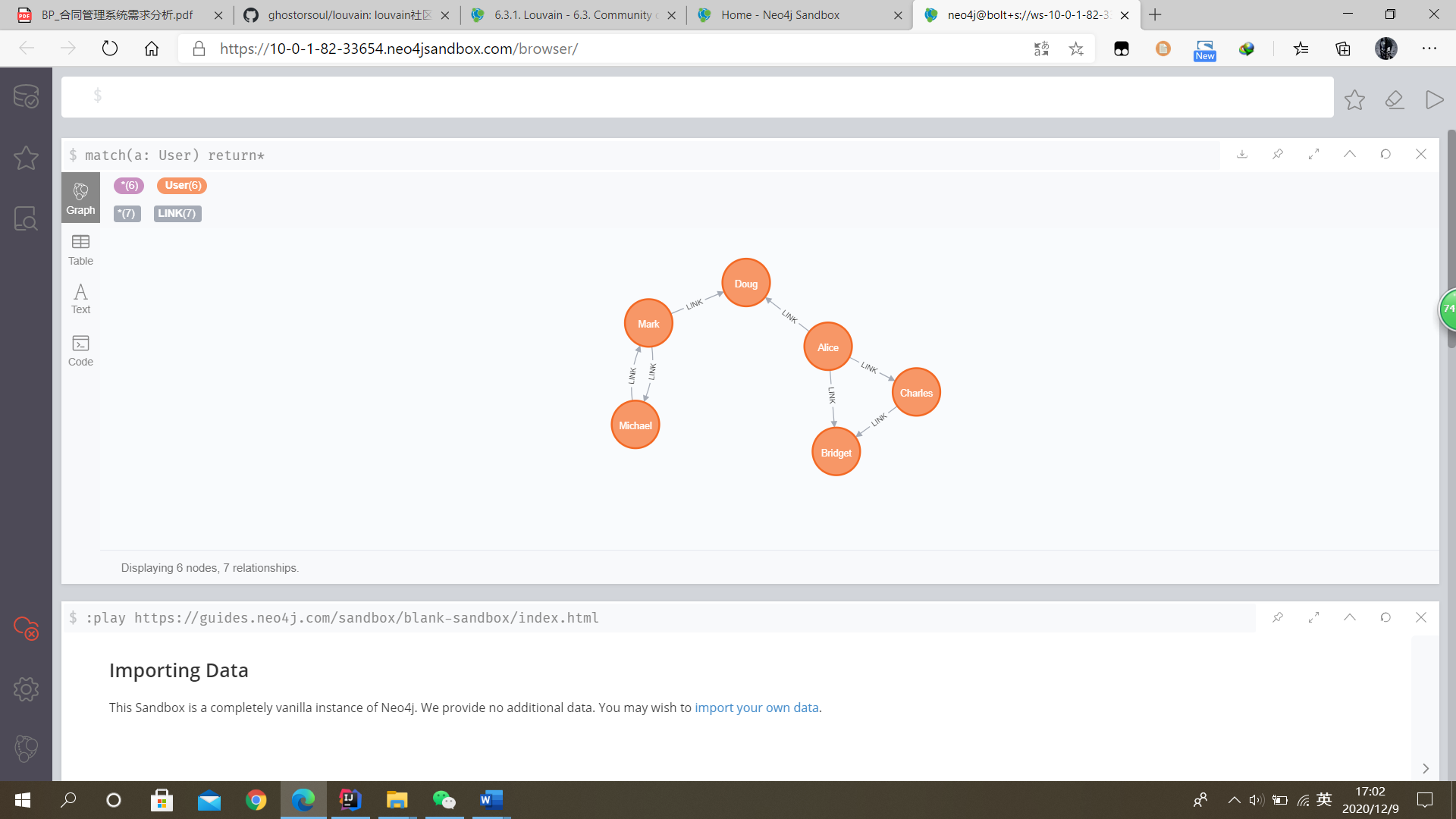
(nMark)-[:LINK {weight: 1}]->(nMichael),

(nMichael)-[:LINK {weight: 1}]->(nMark);



1. 测试数据库操作

match(a: User) return\*



1. 创建Graph

CALL gds.graph.create(

'myGraph',

'User',

{

LINK: {

orientation: 'UNDIRECTED'

}

},

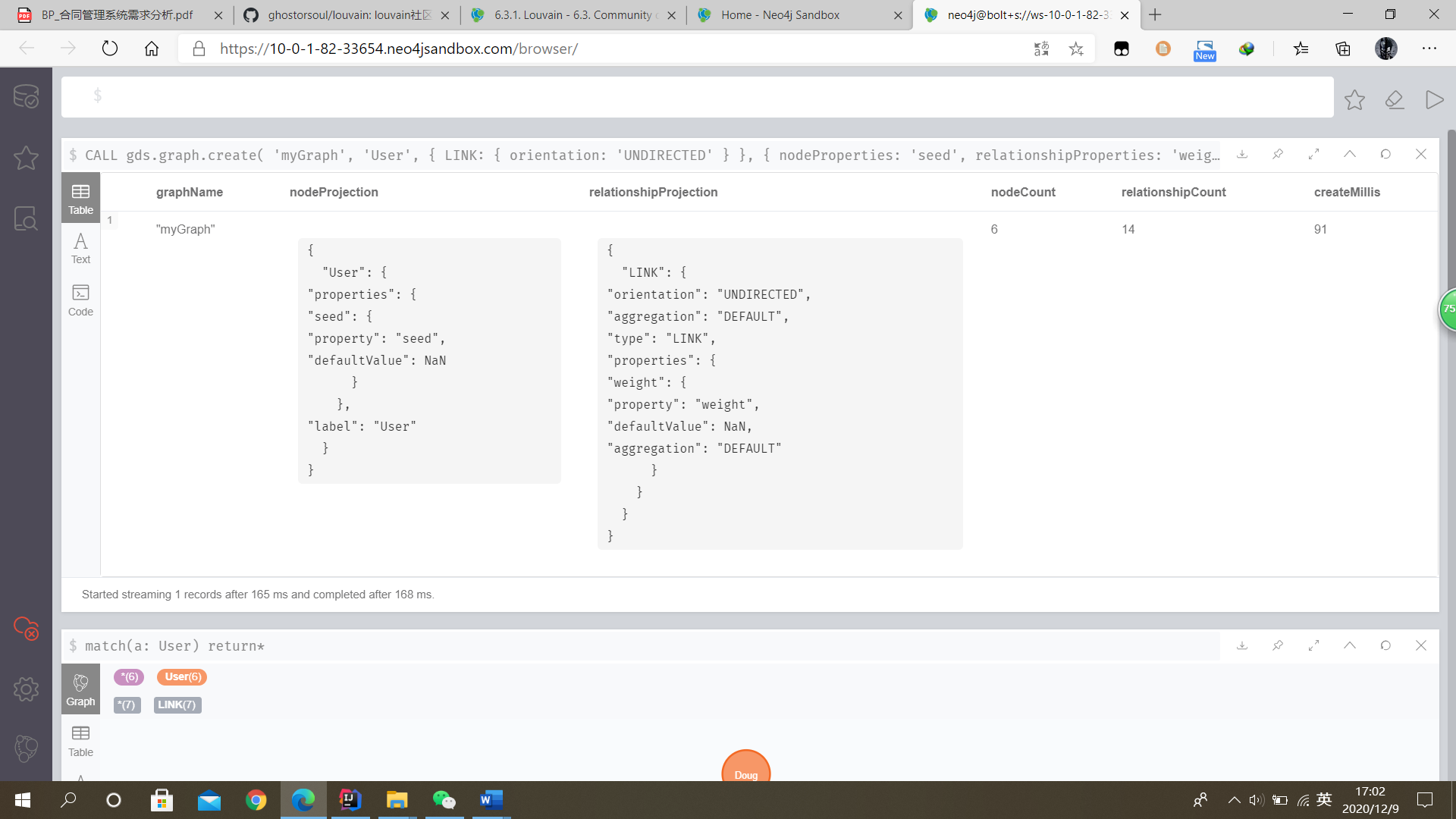
{

nodeProperties: 'seed',

relationshipProperties: 'weight'

}

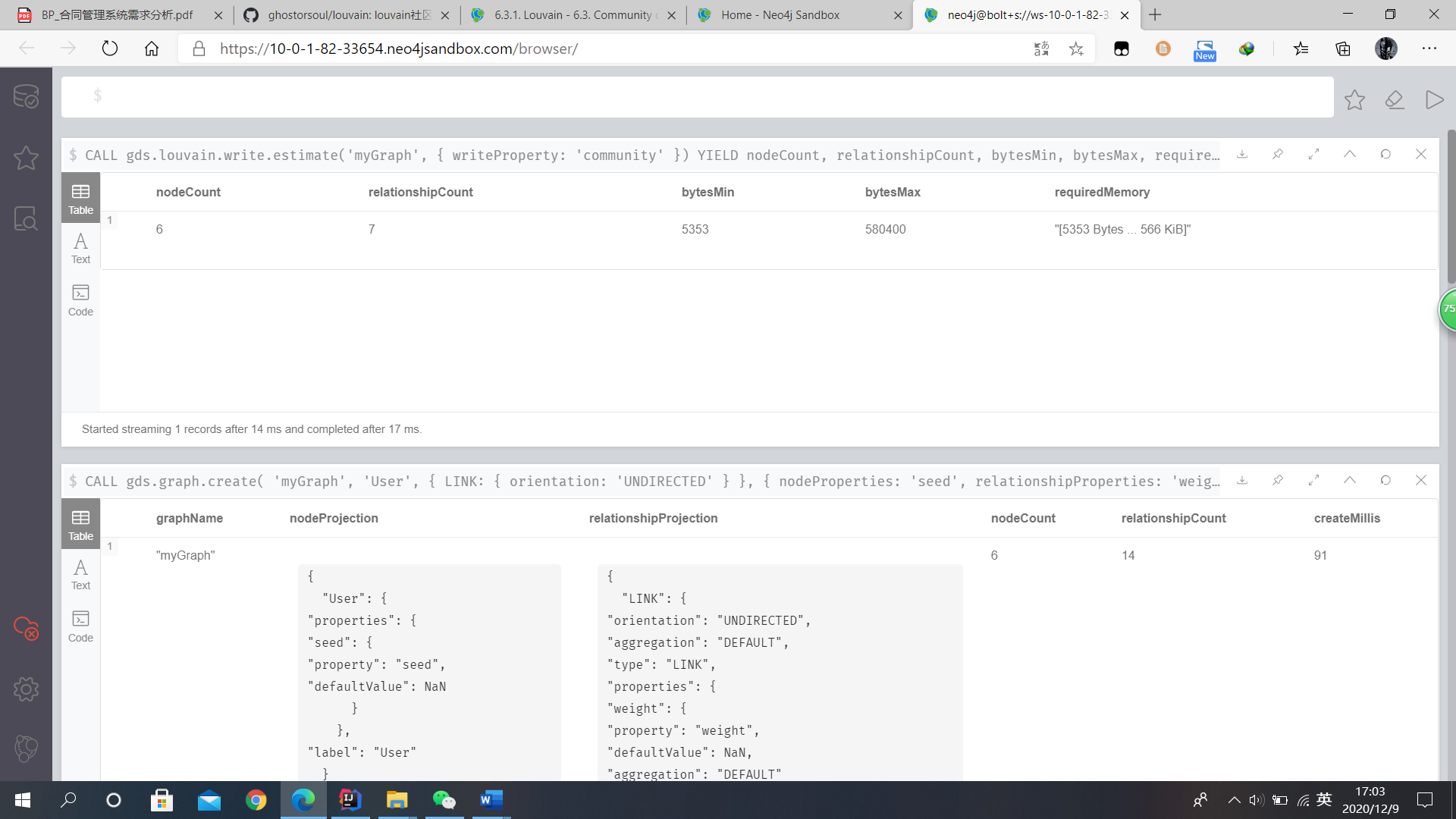
)



1. 评估运行算法的内存需求

CALL gds.louvain.write.estimate('myGraph', { writeProperty: 'community' })

YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory



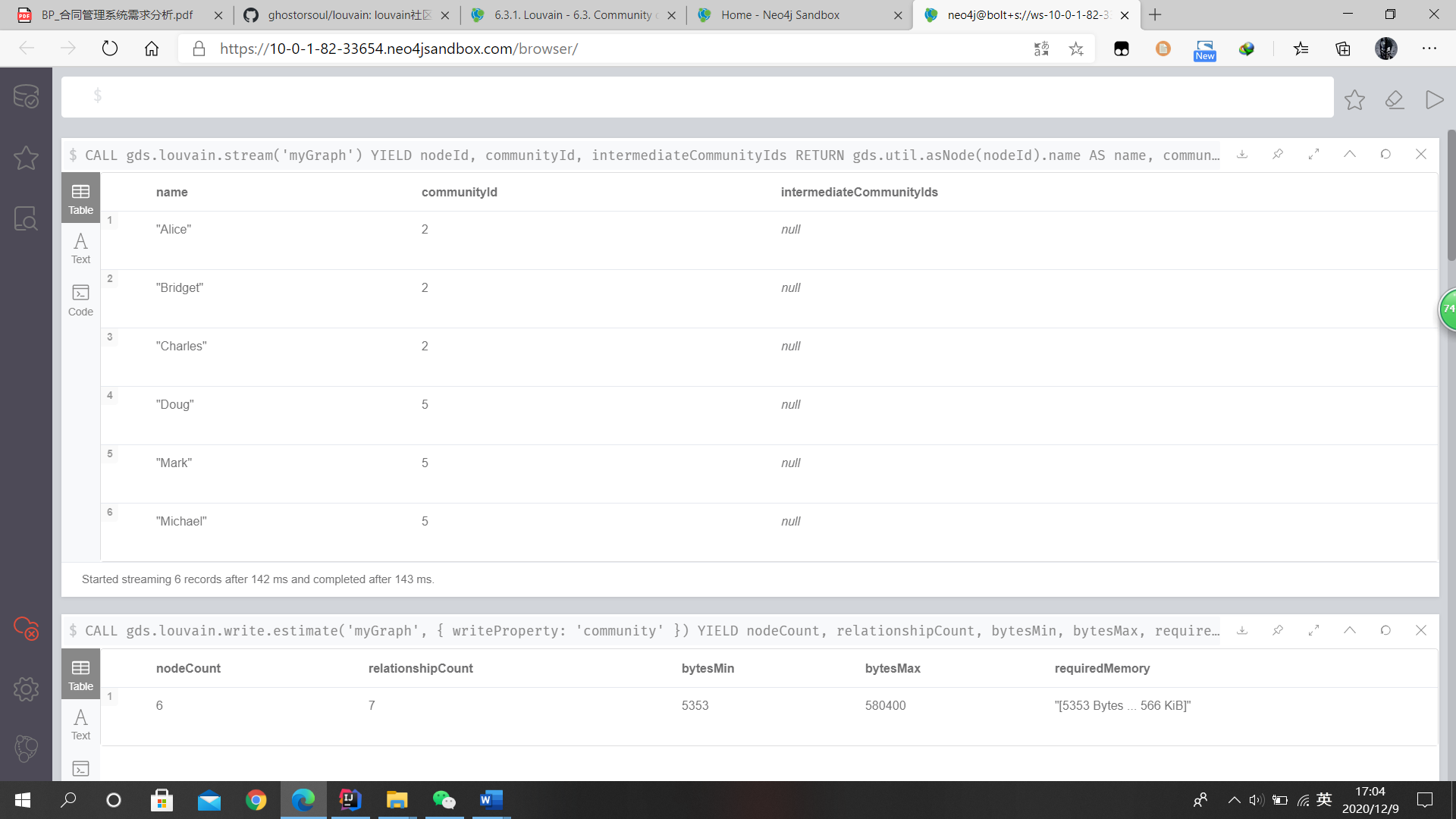
1. 运行Louvain算法并显示结果

CALL gds.louvain.stream('myGraph')

YIELD nodeId, communityId, intermediateCommunityIds

RETURN gds.util.asNode(nodeId).name AS name, communityId, intermediateCommunityIds

ORDER BY name ASC



1. 运行Louvain算法并将结果存储在MyGraph中

CALL gds.louvain.mutate('myGraph', { mutateProperty: 'communityId' })

YIELD communityCount, modularity, modularities

