

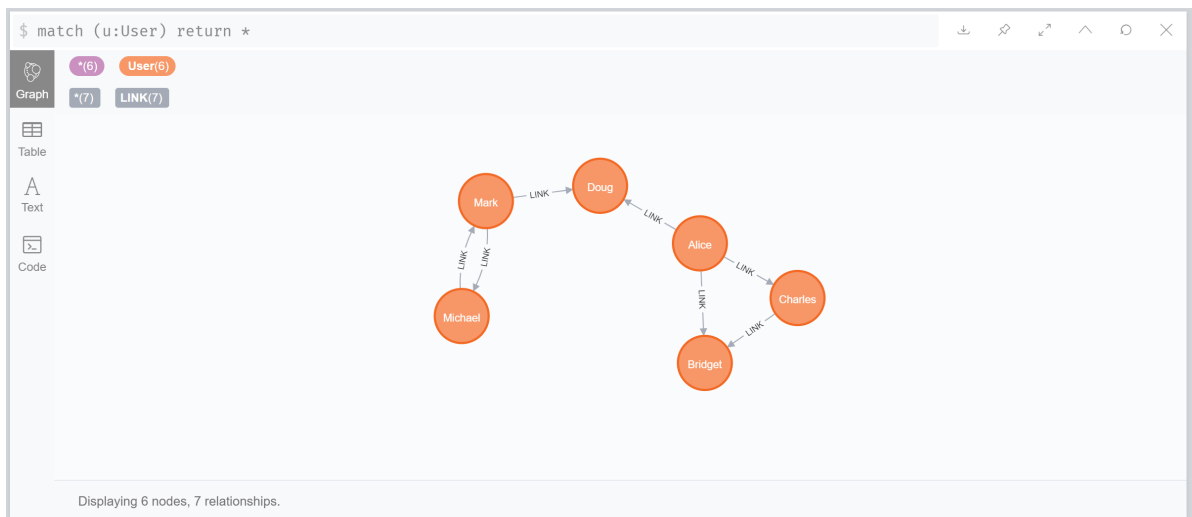
neo4j

导入数据

```
1 CREATE
2   (nAlice:User {name: 'Alice', seed: 42}),
3   (nBridget: User {name: 'Bridget', seed: 42}),
4   (nCharles: User {name: 'Charles', seed: 42}),
5   (nDoug: User {name: 'Doug'}),
6   (nMark: User {name: 'Mark'}),
7   (nMichael: User {name: 'Michael'}),
8   (nAlice)-[:LINK {weight: 1}]->(nBridget),
9   (nAlice)-[:LINK {weight: 1}]->(nCharles),
10  (nCharles)-[:LINK {weight: 1}]->(nBridget),
11  (nAlice)-[:LINK {weight: 5}]->(nDoug),
12  (nMark)-[:LINK {weight: 1}]->(nDoug),
13  (nMark)-[:LINK {weight: 1}]->(nMichael),
14  (nMichael)-[:LINK {weight: 1}]->(nMark);
```

验证导入数据

```
1 match (u:User) return *
```



准备工作

```

1 CALL gds.graph.create(
2   'myGraph',
3   'User',
4   {
5     LINK: {
6       orientation: 'UNDIRECTED'
7     }
8   },
9   {
10    nodeProperties: 'seed',
11    relationshipProperties: 'weight'
12  }
13 )

```

\$ CALL gds.graph.create('myGraph', 'User', { LINK: { orientation: 'UNDIRECTED' } }, { nodePropert...

	graphName	nodeProjection	relationshipProjection	nodeCount	relationshipCount	createMillis
1	"myGraph"	{ "User": { "properties": { "seed": { "property": "seed", "defaultValue": NaN } }, "label": "User" } }	{ "LINK": { "orientation": "UNDIRECTED", "aggregation": "DEFAULT", "type": "LINK", "properties": { "weight": { "property": "weight", "defaultValue": NaN, "aggregation": "DEFAULT" } } } }	6	14	197

评估算法所需资源

```

1 CALL gds.louvain.write.estimate('myGraph', { writeProperty: 'community' })
2 YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory

```

\$ CALL gds.louvain.write.estimate('myGraph', { writeProperty: 'community' }) YIELD nodeCount, rela...

	nodeCount	relationshipCount	bytesMin	bytesMax	requiredMemory
1	6	7	5353	580400	"[5353 Bytes ... 586 KiB]"

Started streaming 1 records after 13 ms and completed after 16 ms.

运行louvain算法

返回流结果

```
1 CALL gds.louvain.stream('myGraph')
2 YIELD nodeId, communityId, intermediateCommunityIds
3 RETURN gds.util.asNode(nodeId).name AS name, communityId,
4         intermediateCommunityIds
5 ORDER BY name ASC
```

The screenshot shows a graph database interface with a sidebar on the left containing icons for Table, Text, and Code. The main area displays the results of a query in a table format. The query is: `$ CALL gds.louvain.stream('myGraph') YIELD nodeId, communityId, intermediateCommunityIds RETURN gds.util.asNode(nodeId).name AS name, communityId, intermediateCommunityIds ORDER BY name ASC`. The results table has three columns: "name", "communityId", and "intermediateCommunityIds". The data rows are:

"name"	"communityId"	"intermediateCommunityIds"
"Alice"	2	null
"Bridget"	2	null
"Charles"	2	null
"Doug"	5	null
"Mark"	5	null
"Michael"	5	null

Below this table, there is another query: `$ CALL gds.louvain.write.estimate('myGraph', { writeProperty: 'community' }) YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory`. The results table for this query has five columns: "nodeCount", "relationshipCount", "bytesMin", "bytesMax", and "requiredMemory". The data row is:

"nodeCount"	"relationshipCount"	"bytesMin"	"bytesMax"	"requiredMemory"
6	7	5353	580400	"[5353 Bytes ... 566 KiB]"

返回社区数

```
1 CALL gds.louvain.stats('myGraph')
2 YIELD communityCount
```

```
CALL gds.louvain.stats('myGraph') YIELD communityCount
```

The screenshot shows a graph database interface with a sidebar on the left containing icons for Table, Text, and Code. The main area displays the result of a query in a table format. The query is: `CALL gds.louvain.stats('myGraph') YIELD communityCount`. The results table has one column: "communityCount". The data row is:

"communityCount"
2

返回模块度

```
1 CALL gds.louvain.mutate('myGraph', { mutateProperty: 'communityId' })
2 YIELD communityCount, modularity, modularities
```

The screenshot shows a graph database interface with a sidebar on the left containing icons for Table, Text, and Code. The main area displays the results of a query in a table format. The query is: `$ CALL gds.louvain.mutate('myGraph', { mutateProperty: 'communityId' }) YIELD communityCount, modularity, modularities`. The results table has three columns: "communityCount", "modularity", and "modularities". The data row is:

"communityCount"	"modularity"	"modularities"
2	0.3571428571428571	[0.3571428571428571]

