COMP8210/COMP7210

Big Data Technologies

Assignment 2

Semester 2, 2024

Name: Rohan Junaid Khan

SID: 47843276

YOUTUBE LINK: https://www.youtube.com/watch?v=89tAwG0oeP0

PART 1: Initial Graph Data Model

Install: APOC & GDS plug-in

Queries:

```
// Constraints for uniqueness
CREATE CONSTRAINT FOR (c:Client) REQUIRE c.id IS UNIQUE;
CREATE CONSTRAINT FOR (s:Seller) REQUIRE s.id IS UNIQUE;
CREATE CONSTRAINT FOR (e:Email) REQUIRE e.address IS UNIQUE;
CREATE CONSTRAINT FOR (p:Phone) REQUIRE p.number IS UNIQUE;
CREATE CONSTRAINT FOR (t:TFN) REQUIRE t.tfn_no IS UNIQUE;

// Indexes for searching effectively
CREATE INDEX FOR (c:Client) ON (c.name);
CREATE INDEX FOR (s:Seller) ON (s.name);


// Create client node with merging email, phone, tfn information as nodes
LOAD CSV WITH HEADERS FROM 'file:///clients.csv' AS row
MERGE (c:Client {id: row.id})
ON CREATE SET c.name = row.name, c.flagged = row.flagged
MERGE (e:Email {address: row.email})
MERGE (p:Phone {number: row.phone})
MERGE (t:TFN {tfn_no: row.tfn})
```

```
MERGE (c)-[:HAS_EMAIL]->(e)
MERGE (c)-[:HAS_PHONE]->(p)
MERGE (c)-[:HAS_TFN]->(t);
```

```
//Create seller node
LOAD CSV WITH HEADERS FROM 'file:///stores.csv' AS row
MERGE (s:Seller {id: row.id})
ON CREATE SET s.name = row.name;
```

```
// Import Transfers and create transaction and transfer node correspondingly
LOAD CSV WITH HEADERS FROM 'file:///xfer.csv' AS row
MATCH (sender:Client {id: row.idFrom}), (receiver:Client {id: row.idTo})
CREATE (sender)-[:PERFORMED]->(t:Transfer {
 amount: toFloat(row.amount),
 time: datetime({epochSeconds: toInteger(row.timeOffset) + 1684665600})
})
MERGE (sender)-[:PERFORMED]->(tx:Transaction {
 from: row.nameFrom,
 to: row.nameTo,
 amount: toFloat(row.amount),
 time: datetime({epochSeconds: toInteger(row.timeOffset) + 1683849600 })
})
ON CREATE SET tx.Type = "transfer"
CREATE (t)-[:TO]->(receiver)
MERGE (tx)-[:TO]->(receiver);
```

```
// Import Purchases and create transaction and purchase node correspondingly
LOAD CSV WITH HEADERS FROM 'file:///purchase.csv' AS row
MATCH (c:Client {id: row.idFrom}), (s:Seller {id: row.idTo})
CREATE (c)-[:PERFORMED]->(p:Purchase {
  amount: toFloat(row.amount),
  time: datetime({epochSeconds: toInteger(row.timeOffset) + 1683849600 })
})
MERGE (c)-[:PERFORMED]->(tx:Transaction {
  from: row.nameFrom,
  to: row.nameTo,
  amount: toFloat(row.amount),
  time: datetime({epochSeconds: toInteger(row.timeOffset) + 1684665600})
})
ON CREATE SET tx.Type = "purchase"
CREATE (p)-[:TO]->(s)
MERGE (tx)-[:TO]->(s);
```

Show graph ontology:

call apoc.meta.graph

or else use,

db.schema.visualization

PART 2: Initial Queries

Problem 1:

// Match purchases and filter based on the substring of the datetime

```
MATCH (c:Client)-[:PERFORMED]->(p:Purchase)
WHERE
  p.time >= datetime("2023-05-12T10:00:00Z") AND p.time < datetime("2023-05-12T14:00:00Z")


// Return the purchase details
RETURN c.name AS Name,SUM(p.amount) AS Total
ORDER BY Total DESC
LIMIT 1
```

Result:

	Name	Total
1	"Logan Adams"	317962.0244923555

Problem 2:

// Calculate outgoing amounts for each client

```
MATCH (client:Client)-[:PERFORMED]->(tx:Transaction)
WITH client, sum(tx.amount) AS totalOutgoing, MAX(tx.amount) AS big_spend


// Calculate incoming amounts for each client
```

```
OPTIONAL MATCH (tx:Transaction)-[:TO]->(client)
WITH client, totalOutgoing, big_spend, sum(tx.amount) AS totalIncoming

// Calculate balance by subtracting outgoing from incoming
WITH client, totalIncoming, totalOutgoing, big_spend, totalIncoming - totalOutgoing AS balance
RETURN client.name AS Name, balance, big_spend
ORDER BY balance ASC
LIMIT 5
```

Result:

Name	balance	big_spend
"Faith Woodward"	-75577230.9491865	10368482.378790302
"Logan Adams"	-42806231.327138394	5376406.330189336
"Arianna Henderson"	-40537938.48631537	10084527.498542838
"Morgan Hunt"	-36538083.61054539	3645443.5263553485
"Ayden Galloway"	-33416803.60337599	5198771.052667163

Problem 3:

```
// Find transfers to clients who then make purchases from Seller 'Woods'
MATCH (client:Client)<-[:TO]-(transfer:Transaction {Type: 'transfer'})
WITH client, SUM(transfer.amount) AS total_xfer

// Find purchases from these clients to Seller 'Woods'
MATCH (client)-[:PERFORMED]->(purchase:Transaction)-[:TO]->(seller:Seller {name: 'Woods'})
WITH client, total_xfer, SUM(purchase.amount) AS total_purchase

// Calculate the percentage of received funds spent at 'Woods'
```

```
WITH client, total_xfer, total_purchase,
    (total_purchase * 100.0 / total_xfer) AS percentage

// Filter clients where at least 5% of received transfers are spent on purchases from 'Woods'
WHERE percentage >= 5

// Return relevant information
RETURN client.name AS Name,
      total_xfer,
      total_purchase,
      percentage
```

Result:

Name	total_xfer	total_purchase	percentage
"Jackson Lambert"	1955067.8469253501	107104.04902677977	5.478277861058256

Problem 4:

```
//  Match clients and their transactions, order and collect, and create relationships
MATCH (c:Client)-[:PERFORMED]->(t:Transaction)
WHERE t.from = c.name
WITH c, t
ORDER BY t.time
WITH c, collect(t) AS transactions
WHERE size(transactions) > 1

// Establish "first_tx" and "last_tx" relationships
WITH c, transactions, head(transactions) AS firstTx, last(transactions) AS lastTx
MERGE (c)-[:first_tx]->(firstTx)
MERGE (c)-[:last_tx]->(lastTx)
```

```
// Chain transactions with "next" relationships and return results
WITH transactions
UNWIND range(0, size(transactions) - 2) AS i
WITH transactions[i] AS currentTransaction, transactions[i+1] AS nextTransaction
MERGE (currentTransaction)-[:next]->(nextTransaction)



// Delete the previous relationship of client with transactions
MATCH (c:Client)-[r:PERFORMED]->(t:Transaction)
DELETE r

// Check one client node to see the overall result of new relationships (Figure 2)
match (c:Client {name: "Noah Miller"}), (t:Transaction {from: "Noah Miller"}) return c,t
```

Result:

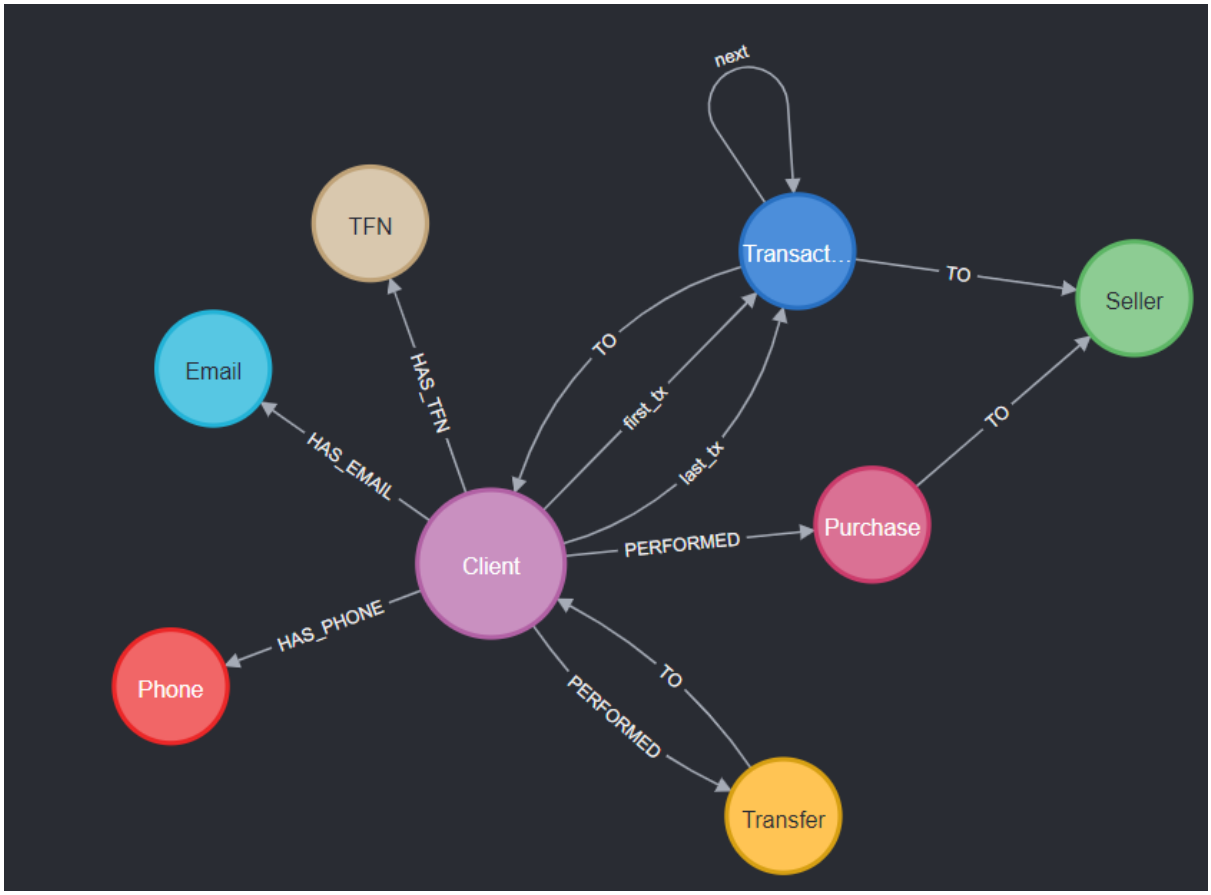


Figure 1: New graph schema after changing relation between client and transactions.

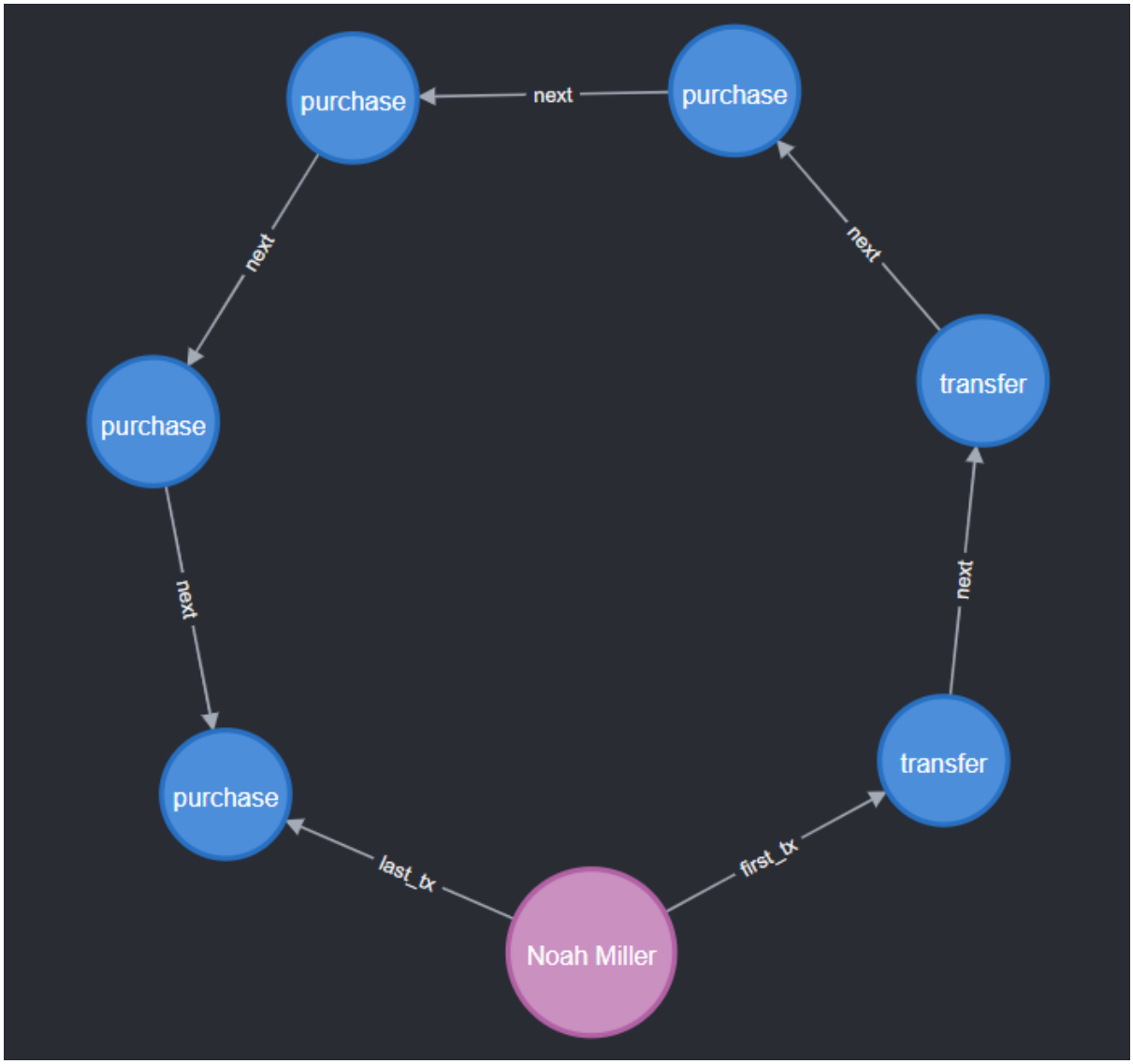


Figure 2: Relation of transactions of client node "Noah Miller"

Part 3: Graph Data Science

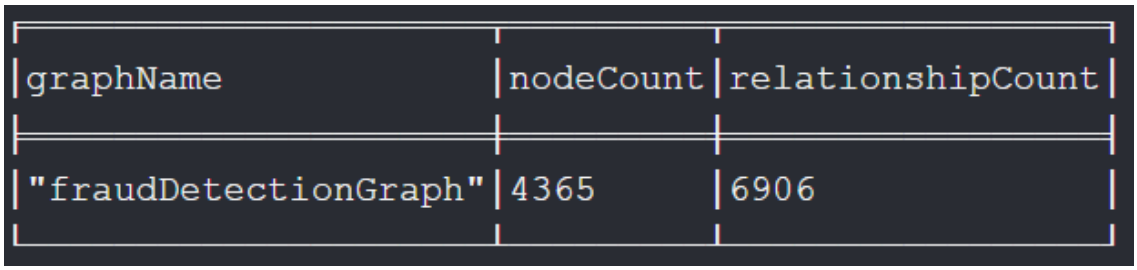
Part A:

i)

```
// Create graph projection for fraud detection
CALL gds.graph.project(
 'fraudDetectionGraph',
 ['Client', 'Email', 'Phone', 'TFN'],  // Including nodes
 {
  HAS_EMAIL: {
```

```
    type: 'HAS_EMAIL',
    orientation: 'UNDIRECTED'
  },
  HAS_PHONE: {
    type: 'HAS_PHONE',
    orientation: 'UNDIRECTED'
  },
  HAS_TFN: {
    type: 'HAS_TFN',
    orientation: 'UNDIRECTED'
  }
 }
)
YIELD graphName, nodeCount, relationshipCount
RETURN graphName, nodeCount, relationshipCount
```

Output:



graphName	nodeCount	relationshipCount
"fraudDetectionGraph"	4365	6906

```
// Applying label propagation algorithm
CALL gds.labelPropagation.write('fraudDetectionGraph', { writeProperty: 'communityId' })
```

ii)

// Identify groups with atleast 5 members

MATCH (c:Client)

WHERE c.communityId IS NOT NULL

WITH c.communityId AS groupId, COUNT(c) AS groupSize

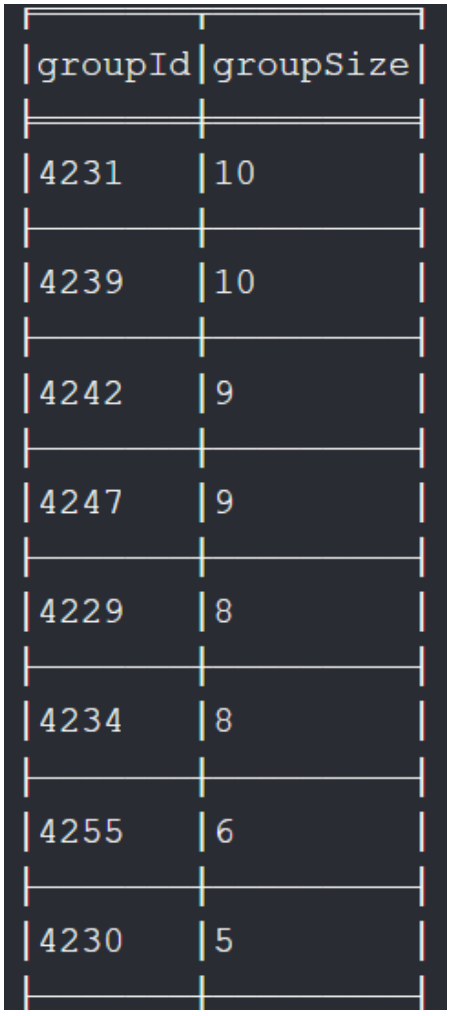
WHERE groupSize >=5

With groupId, groupSize

ORDER BY groupSize DESC

RETURN groupId, groupSize

Output:



groupId	groupSize
4231	10
4239	10
4242	9
4247	9
4229	8
4234	8
4255	6
4230	5

// Assigning groupId in the main dataset

MATCH (c:Client)

WHERE c.communityId IS NOT NULL

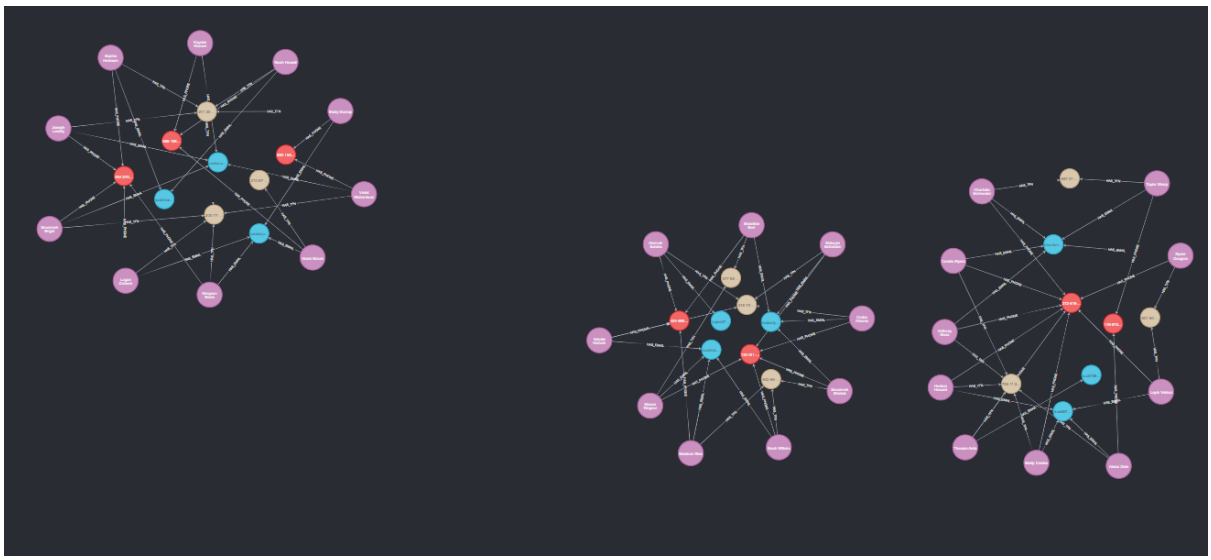
WITH c.communityId AS groupId, COUNT(c) AS groupSize

```
WHERE groupSize >=5
WITH groupId
MATCH (c:Client {communityId: groupId})
SET c.groupId = groupId
```

iii)

```
// Visualization of the largest groups
MATCH (c:Client)-[r:HAS_EMAIL|HAS_TFN|HAS_PHONE]->(id)
WHERE c.communityId IS NOT NULL
AND c.communityId IN [4231,4239,4242]
RETURN c AS Client, id AS SharedIdentifier, r AS Relation, c.communityId AS groupId
ORDER BY groupId
```

Output:



Part B:

i)

```
// Transfers between fraud group members and other group members
MATCH (c1:Client)-[:PERFORMED]->(t:Transfer)-[:TO]->(c2:Client)
WHERE c1.groupId IS NOT NULL AND (c2.groupId IS NULL OR c1.groupId <> c2.groupId)
```

RETURN c1.name AS FraudMember, c2.name AS NonfraudClient, t.amount AS Amount

ORDER BY Amount DESC

Output:

FraudMember	NonfraudClient	Amount
"Anthony Pacheco"	"Addison Mueller"	696589.5991861614
"Joseph Landry"	"Juan Williams"	665399.6761444275
"Maya Lowery"	"Andrew Adkins"	525038.2089777053
"Abigail Hardin"	"Grace Dickerson"	199636.1402714078
"Kennedy Kline"	"Andrew Adkins"	183166.10481309047

ii)

```
// Projecting graph for recent larger group
CALL gds.graph.project.cypher(
   'largerFraudGroupGraph',
   'MATCH (c:Client) RETURN id(c) AS id',
   'MATCH (c1:Client)-[:PERFORMED]->(t:Transfer)-   [:TO]->(c2:Client)
   WHERE c1.groupId IS NOT NULL AND (c2.groupId IS NULL OR c1.groupId <> c2.groupId)
   RETURN id(t) AS id, id(c1) AS source, id(c2) AS target',
   {validateRelationships: false}
)

YIELD graphName, nodeCount, relationshipCount
RETURN graphName, nodeCount, relationshipCount
```

iii)

```
// Identifying central players using PageRank algorithm
CALL gds.pageRank.stream('largerFraudGroupGraph')
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS node, score
WHERE node:Client
SET node.keySuspect = (score > 0.5)
RETURN node.name AS ClientName, score, node.keySuspect
ORDER BY score DESC
```

Output:

ClientName	score	node.keySuspect
"Juan Williams"	1.4249999999999998	true
"Addison Mueller"	1.4249999999999998	true
"Thomas Spence"	1.4249999999999998	true
"Tristan Sosa"	1.2974999999999999	true
"Hannah Byers"	1.2974999999999999	true
"Andrew Adkins"	1.17	true

iv)

```
// Visualizing the result
MATCH (l:Client)-[p:PERFORMED]->(t:Transfer)-[:TO]->(suspect:Client)
WHERE suspect.keySuspect = true
RETURN l,t,suspect
```

Output: