**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:**

callapoc.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:***

A black rectangular object with a white line

Description automatically generated

***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:***

A screenshot of a computer

Description automatically generated

***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:***

A screen shot of a computer

Description automatically generated

***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:***

A diagram of a company

Description automatically generated

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

A diagram of a diagram

Description automatically generated

Figure : 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:**

**A screenshot of a computer

Description automatically generated**

// 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:**

**A black background with white lines and numbers

Description automatically generated**

// 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:**

**A computer screen shot of a network

Description automatically generated**

**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:**

**A screenshot of a computer screen

Description automatically generated**

**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:**

**A screenshot of a computer program

Description automatically generated**

**iv)**

// Visualizing the result

MATCH (l:Client)-[p:PERFORMED]->(t:Transfer)-[:TO]->(suspect:Client)

WHERE suspect.keySuspect = true

RETURN l,t,suspect

**Output:**

**A pattern of yellow and purple circles

Description automatically generated**