**APPENDIX: CODE FOR NEO4J PROJECT GRAPH KNOWLEDGE**

**Step 1: Creating Indexes**

Creating indexes in Neo4j improves query performance, especially when using the MERGE operation to find or create nodes. The following code creates indexes on key properties for each type of node.

cypher

// Create an index on the `identifier` property of the RecordID node

CREATE INDEX FOR (n:RecordID) ON (n.identifier);

// Create an index on the `state` property of the Location node

CREATE INDEX FOR (n:Location) ON (n.state);

// Create an index on the `povertyStatus` property of the PovertyStatus node

CREATE INDEX FOR (n:PovertyStatus) ON (n.povertyStatus);

// Create an index on the `race` property of the Race node

CREATE INDEX FOR (n:Race) ON (n.race);

// Create an index on the `incomerange` property of the Income node

CREATE INDEX FOR (n:Income) ON (n.incomerange);

// Create an index on the `gender` property of the Gender node

CREATE INDEX FOR (n:Gender) ON (n.gender);

// Create an index on the `maritalstatus` property of the MaritalStatus node

CREATE INDEX FOR (n:MaritalStatus) ON (n.maritalstatus);

// Create an index on the `familysize` property of the HouseholdSize node

CREATE INDEX FOR (n:HouseholdSize) ON (n.familysize);

These indexes improve performance for MERGE operations on RecordID, Location, PovertyStatus, Race, Income, and Gender nodes.

**Step 2: Load CSV and Create Individual Nodes**

This step loads data from the Poverty\_in\_US\_coded.csv file and creates unique nodes for each entity (RecordID, Location, PovertyStatus, Race, Income, and Gender). The MERGE command ensures that each node is unique.

cypher

:auto LOAD CSV WITH HEADERS FROM "file:///Poverty\_in\_US\_coded.csv" AS row

CALL { WITH row MERGE (:RecordID {identifier: row.RecordID}) } IN TRANSACTIONS OF 3000 ROWS;

:auto LOAD CSV WITH HEADERS FROM "file:///Poverty\_in\_US\_coded.csv" AS row

CALL { WITH row MERGE (:Location {state: row.Location}) } IN TRANSACTIONS OF 3000 ROWS;

:auto LOAD CSV WITH HEADERS FROM "file:///Poverty\_in\_US\_coded.csv" AS row

CALL { WITH row MERGE (:PovertyStatus {povertyStatus: row.PovertyStatus}) } IN TRANSACTIONS OF 3000 ROWS;

:auto LOAD CSV WITH HEADERS FROM "file:///Poverty\_in\_US\_coded.csv" AS row

CALL { WITH row MERGE (:Race {race: row.Race}) } IN TRANSACTIONS OF 3000 ROWS;

:auto LOAD CSV WITH HEADERS FROM "file:///Poverty\_in\_US\_coded.csv" AS row

CALL { WITH row MERGE (:Income {incomerange: row.Income}) } IN TRANSACTIONS OF 3000 ROWS;

:auto LOAD CSV WITH HEADERS FROM "file:///Poverty\_in\_US\_coded.csv" AS row

CALL { WITH row MERGE (:Gender {gender: row.Gender}) } IN TRANSACTIONS OF 3000 ROWS;

:auto LOAD CSV WITH HEADERS FROM "file:///Poverty\_in\_US\_coded.csv" AS row

CALL { WITH row MERGE (:MaritalStatus {maritalstatus: row.MaritalStatus}) } IN TRANSACTIONS OF 3000 ROWS;

:auto LOAD CSV WITH HEADERS FROM "file:///Poverty\_in\_US\_coded.csv" AS row

CALL { WITH row MERGE (:HouseholdSize {familysize: row.HouseholdSize}) } IN TRANSACTIONS OF 3000 ROWS;

**Explanation of Each Block**

* Each MERGE command ensures that only a single node is created for each unique value in the columns RecordID, Location, PovertyStatus, Race, Income, and Gender.
* The IN TRANSACTIONS OF 3000 ROWS clause processes the data in batches of 3000 rows, which is useful for handling large datasets without overloading the system.

**Step 3: Creating Nodes and Relationships from CSV Data**

This step creates RecordID, Location, Race, Education, PovertyStatus, Gender, and Income nodes, and establishes relationships between the RecordID node and other nodes.

cypher

:auto LOAD CSV WITH HEADERS FROM "file:///Poverty\_in\_US\_coded.csv" AS row

CALL {

WITH row

MERGE (person:RecordID {identifier: row.RecordID})

MERGE (location:Location {state: row.Location})

MERGE (race:Race {ethnicity: row.Race})

MERGE (education:Education {level: row.Education})

MERGE (status:PovertyStatus {status: CASE WHEN row.PovertyStatus IN ['Poor', 'Not Poor'] THEN row.PovertyStatus ELSE 'Unknown' END})

MERGE (gender:Gender {gender: row.Gender})

MERGE (income:Income {incomeRange: row.Income})

   MERGE (maritalstatus:MaritalStatus {maritalstatus: row.MaritalStatus})

   MERGE (familysize:HouseholdSize {familysize: row.HouseholdSize})

MERGE (person)-[:LIVES\_IN]->(location)

MERGE (person)-[:HAS\_RACE]->(race)

MERGE (person)-[:HAS\_EDUCATION]->(education)

MERGE (person)-[:HAS\_STATUS]->(status)

MERGE (person)-[:HAS\_GENDER]->(gender)

MERGE (person)-[:HAS\_INCOME]->(income)

   MERGE (person)-[:HAS\_MARITAL\_STATUS]->(maritalstatus)

   MERGE (person)-[:HAS\_FAMILY\_SIZE]->(familysize)

} IN TRANSACTIONS OF 3000 ROWS;

To eliminate nodes values

MATCH (n:HouseholdSize) DETACH DELETE n;

**Explanation of Each Line**

* **Create RecordID Node**:

cypher

Copiar código

MERGE (person:RecordID {identifier: row.RecordID})

Ensures that each person has a unique RecordID node.

* **Create Location Node and Relationship**:

cypher

Copiar código

MERGE (location:Location {state: row.Location})

MERGE (person)-[:LIVES\_IN]->(location)

Creates a Location node (if it doesn't already exist) and establishes the LIVES\_IN relationship.

* **Create Race Node and Relationship**:

cypher

Copiar código

MERGE (race:Race {ethnicity: row.Race})

MERGE (person)-[:HAS\_RACE]->(race)

Creates a Race node and establishes the HAS\_RACE relationship.

* **Create Education Node and Relationship**:

cypher

Copiar código

MERGE (education:Education {level: row.Education})

MERGE (person)-[:HAS\_EDUCATION]->(education)

Creates an Education node and establishes the HAS\_EDUCATION relationship.

* **Create PovertyStatus Node with Default Value and Relationship**:

cypher

Copiar código

MERGE (status:PovertyStatus {status: CASE WHEN row.PovertyStatus IN ['Poor', 'Not Poor'] THEN row.PovertyStatus ELSE 'Unknown' END})

MERGE (person)-[:HAS\_STATUS]->(status)

Assigns a default value of Unknown if PovertyStatus is not Poor or Not Poor.

* **Create Gender Node and Relationship**:

cypher

Copiar código

MERGE (gender:Gender {gender: row.Gender})

MERGE (person)-[:HAS\_GENDER]->(gender)

Creates a Gender node and establishes the HAS\_GENDER relationship.

* **Create Income Node and Relationship**:

cypher

Copiar código

MERGE (income:Income {incomeRange: row.Income})

MERGE (person)-[:HAS\_INCOME]->(income)

Creates an Income node and establishes the HAS\_INCOME relationship.

**Step 4: Verifying Relationships (Optional)**

To ensure that each RecordID has the required relationships, you can run verification queries as shown in the image provided.

cypher

// Check for nodes missing the LIVES\_IN relationship

MATCH (person:RecordID) WHERE NOT (person)-[:LIVES\_IN]->() RETURN person.identifier AS MissingLIVES\_IN LIMIT 10;

// Check for nodes missing the HAS\_RACE relationship

MATCH (person:RecordID) WHERE NOT (person)-[:HAS\_RACE]->() RETURN person.identifier AS MissingHAS\_RACE LIMIT 10;

// Check for nodes missing the HAS\_EDUCATION relationship

MATCH (person:RecordID) WHERE NOT (person)-[:HAS\_EDUCATION]->() RETURN person.identifier AS MissingHAS\_EDUCATION LIMIT 10;

// Check for nodes missing the HAS\_STATUS relationship

MATCH (person:RecordID) WHERE NOT (person)-[:HAS\_STATUS]->() RETURN person.identifier AS MissingHAS\_STATUS LIMIT 10;

// Check for nodes missing the HAS\_GENDER relationship

MATCH (person:RecordID) WHERE NOT (person)-[:HAS\_GENDER]->() RETURN person.identifier AS MissingHAS\_GENDER LIMIT 10;

// Check for nodes missing the HAS\_INCOME relationship

MATCH (person:RecordID) WHERE NOT (person)-[:HAS\_INCOME]->() RETURN person.identifier AS MissingHAS\_INCOME LIMIT 10;

These queries return RecordID nodes that are missing specific relationships, helping identify any data gaps or import issues.

**Creating communities**

CALL gds.graph.project(

  'mytrial2',

  ['RecordID', 'Location', 'PovertyStatus', 'Race', 'Income', 'Gender'],  // Focus on core node types

  {

    HAS\_RACE: {},

    HAS\_EDUCATION: {},

    HAS\_GENDER: {},

    HAS\_STATUS: {},

    LIVES\_IN: {},

    HAS\_INCOME: {}

  }

)

CALL gds.louvain.write(

  'mytrial2',

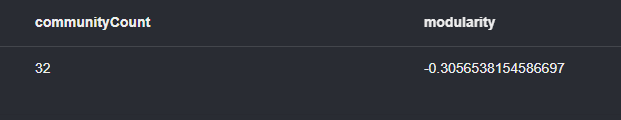
  {

    writeProperty: 'communityId'  // Property name to store community IDs

  }

)

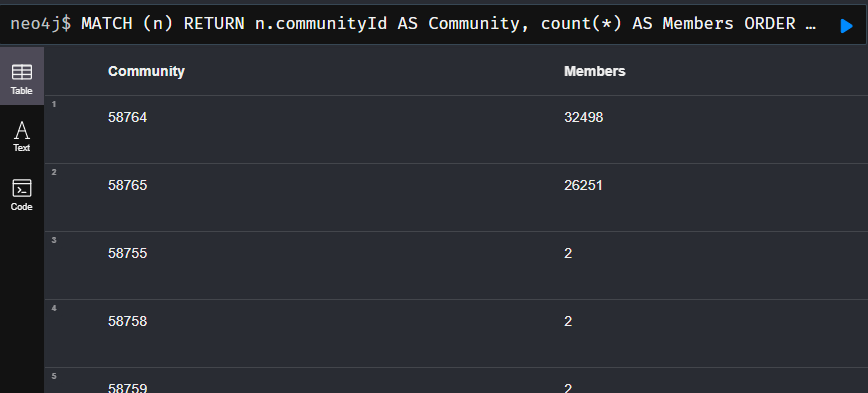
YIELD communityCount, modularity;



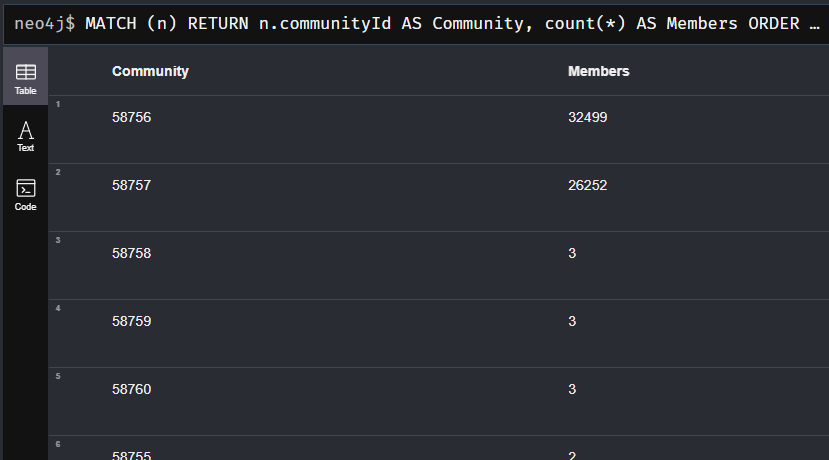
MATCH (n)

RETURN n.communityId AS Community, count(\*) AS Members

ORDER BY Members DESC;



Trial 3 without income



**To clean communityId**

MATCH (n)

REMOVE n.communityId;

**Create the node and relationship of community**

MATCH (n) WITH DISTINCT n.communityId AS communityId CREATE (:Community {id: communityId});

MATCH (n), (c:Community) WHERE n.communityId = c.id MERGE (n)-[:BELONGS\_TO]->(c);

**Change the name of the communities**

MATCH (n)

WITH DISTINCT n.communityId AS communityId

ORDER BY communityId

WITH collect(communityId) AS communityIds

UNWIND range(0, size(communityIds) - 1) AS idx

WITH communityIds[idx] AS communityId, idx + 1 AS newId

MERGE (c:Community {id: communityId})

SET c.name = toString(newId);

MATCH (n)-[r:BELONGS\_TO]->(c:Community) SET r.communityName = c.name;

**Similarities**

**Creating a Projected Graph**

CALL gds.graph.project(

  'povertyGraph',

  ['RecordID', 'Location', 'Race', 'Gender', 'PovertyStatus','Education','HouseholdSize', 'MaritalStatus' ],

  {

    LIVES\_IN: {},

    HAS\_RACE: {},

    HAS\_FAMILY\_SIZE: {},

    HAS\_EDUCATION: {},

    HAS\_MARITAL\_STATUS: {},

    HAS\_GENDER: {},

    HAS\_STATUS: {}

  }

)

**Running Node Similarity and Streaming Results**

CALL gds.nodeSimilarity.stream('povertyGraph', {

  relationshipTypes: ['HAS\_RACE', 'HAS\_GENDER', 'HAS\_STATUS', 'HAS\_FAMILY\_SIZE', 'HAS\_EDUCATION', 'HAS\_MARITAL\_STATUS']

})

YIELD node1, node2, similarity

RETURN gds.util.asNode(node1).identifier AS RecordID1,

       gds.util.asNode(node2).identifier AS RecordID2,

       similarity

ORDER BY similarity DESC

LIMIT 10;

**Writing Node Similarity Relationships to the Database**

CALL gds.nodeSimilarity.write('povertyGraph', {

writeRelationshipType: 'SIMILAR\_TO',

writeProperty: 'similarity'

})

YIELD nodesCompared, relationshipsWritten;

**TABLES**

**Comunities**

Interfaz de usuario gráfica, Aplicación, Correo electrónico

Descripción generada automáticamente

// Step 1: Match nodes with their Community and PovertyStatus

MATCH (person:RecordID)-[:BELONGS\_TO]->(community:Community)

MATCH (person)-[:HAS\_STATUS]->(status:PovertyStatus)

// Step 2: Count the number of people for each combination of Community and PovertyStatus

WITH

community.name AS CommunityName,

status.status AS PovertyStatus,

COUNT(person) AS StatusCount

// Step 3: Calculate the total number of people in each Community

WITH CommunityName, PovertyStatus, StatusCount

MATCH (person:RecordID)-[:BELONGS\_TO]->(community:Community {name: CommunityName})

WITH CommunityName, PovertyStatus, StatusCount, COUNT(person) AS TotalPeopleInCommunity

// Step 4: Calculate percentages

RETURN

CommunityName,

PovertyStatus,

ROUND((StatusCount \* 1.0 / TotalPeopleInCommunity) \* 100, 2) AS Percentage

ORDER BY CommunityName, PovertyStatus;

**TABLAS: ANALYSIS**

**What is the educational level of the majority of people living in poverty?**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

// Step 1: Match RecordID nodes with their Income and Education levels

MATCH (person:RecordID)-[:HAS\_INCOME]->(income:Income)

MATCH (person)-[:HAS\_EDUCATION]->(education:Education)

// Step 2: Count the number of people for each combination of IncomeRange and EducationLevel

WITH

income.incomeRange AS IncomeRange,

education.level AS EducationLevel,

COUNT(person) AS EducationCount

// Step 3: Calculate the total number of people for each IncomeRange

WITH IncomeRange, EducationLevel, EducationCount

MATCH (person:RecordID)-[:HAS\_INCOME]->(income:Income {incomeRange: IncomeRange})

WITH IncomeRange, EducationLevel, EducationCount, COUNT(person) AS TotalPeopleInIncomeRange

// Step 4: Calculate percentages

RETURN

IncomeRange,

EducationLevel,

ROUND((EducationCount \* 1.0 / TotalPeopleInIncomeRange) \* 100, 2) AS Percentage

ORDER BY IncomeRange, EducationLevel;

**What is the educational level of the majority of people living in poverty?**

Tabla

Descripción generada automáticamente

// Step 1: Match RecordID nodes with their PovertyStatus and Education levels

MATCH (person:RecordID)-[:HAS\_STATUS]->(status:PovertyStatus)

MATCH (person)-[:HAS\_EDUCATION]->(education:Education)

// Step 2: Count the number of people for each combination of PovertyStatus and EducationLevel

WITH

  status.status AS PovertyStatus,

  education.level AS EducationLevel,

  COUNT(person) AS EducationCount

// Step 3: Calculate the total number of people for each PovertyStatus

WITH PovertyStatus, EducationLevel, EducationCount

MATCH (person:RecordID)-[:HAS\_STATUS]->(status:PovertyStatus {status: PovertyStatus})

WITH PovertyStatus, EducationLevel, EducationCount, COUNT(person) AS TotalPeopleInPovertyStatus

// Step 4: Calculate percentages

RETURN

  PovertyStatus,

  EducationLevel,

  ROUND((EducationCount \* 1.0 / TotalPeopleInPovertyStatus) \* 100, 2) AS Percentage

ORDER BY PovertyStatus, EducationLevel;

**Are unmarried or unpartnered individuals more likely to experience poverty?**

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

// Step 1: Match RecordID nodes with their PovertyStatus and MaritalStatus

MATCH (person:RecordID)-[:HAS\_STATUS]->(status:PovertyStatus)

MATCH (person)-[:HAS\_MARITAL\_STATUS]->(maritalStatus:MaritalStatus)

// Step 2: Count the number of people for each combination of PovertyStatus and MaritalStatus

WITH

status.status AS PovertyStatus,

maritalStatus.maritalstatus AS MaritalStatus,

COUNT(person) AS MaritalStatusCount

// Step 3: Calculate the total number of people for each PovertyStatus

WITH PovertyStatus, MaritalStatus, MaritalStatusCount

MATCH (person:RecordID)-[:HAS\_STATUS]->(status:PovertyStatus {status: PovertyStatus})

WITH PovertyStatus, MaritalStatus, MaritalStatusCount, COUNT(person) AS TotalPeopleInPovertyStatus

// Step 4: Calculate percentages

RETURN

PovertyStatus,

MaritalStatus,

ROUND((MaritalStatusCount \* 1.0 / TotalPeopleInPovertyStatus) \* 100, 2) AS Percentage

ORDER BY PovertyStatus, MaritalStatus;

**Is there a correlation between being African American and the likelihood of experiencing poverty?**

Captura de pantalla de computadora

Descripción generada automáticamente

// Step 1: Count the number of people of each race in each community

MATCH (person:RecordID)-[:BELONGS\_TO]->(community:Community)

MATCH (person)-[:HAS\_RACE]->(race:Race)

WITH community.name AS CommunityName, race.ethnicity AS Race, COUNT(person) AS RaceCount

// Step 2: Calculate the total number of people in each community

WITH CommunityName, Race, RaceCount

MATCH (person:RecordID)-[:BELONGS\_TO]->(c:Community {name: CommunityName})

WITH CommunityName, Race, RaceCount, COUNT(person) AS TotalPeopleInCommunity

// Step 3: Calculate the percentage for each race within the community

RETURN

CommunityName,

Race,

ROUND((RaceCount \* 1.0 / TotalPeopleInCommunity) \* 100, 2) AS Percentage

ORDER BY CommunityName, Race;

**Does gender influence the probability of experiencing poverty?**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

// Step 1: Match RecordID nodes with their PovertyStatus and Gender

MATCH (person:RecordID)-[:HAS\_STATUS]->(status:PovertyStatus)

MATCH (person)-[:HAS\_GENDER]->(gender:Gender)

// Step 2: Count the number of people for each PovertyStatus and Gender

WITH status.status AS PovertyStatus, gender.gender AS Gender, COUNT(person) AS GenderCount

// Step 3: Calculate the total number of people for each PovertyStatus

WITH PovertyStatus, Gender, GenderCount

MATCH (person:RecordID)-[:HAS\_STATUS]->(status:PovertyStatus {status: PovertyStatus})

WITH PovertyStatus, Gender, GenderCount, COUNT(person) AS TotalCount

// Step 4: Calculate percentages for each gender in each PovertyStatus

RETURN

  PovertyStatus,

  Gender,

  ROUND((GenderCount \* 1.0 / TotalCount) \* 100, 2) AS Percentage

ORDER BY PovertyStatus, Gender;

**How is family size related to the likelihood of experiencing poverty?**

Tabla

Descripción generada automáticamente con confianza media

// Step 1: Match RecordID nodes with their PovertyStatus and HouseholdSize

MATCH (person:RecordID)-[:HAS\_STATUS]->(status:PovertyStatus)

MATCH (person)-[:HAS\_FAMILY\_SIZE]->(household:HouseholdSize)

// Step 2: Count the number of people for each PovertyStatus and HouseholdSize

WITH status.status AS PovertyStatus, household.familysize AS HouseholdSize, COUNT(person) AS HouseholdCount

// Step 3: Calculate the total number of people for each PovertyStatus

WITH PovertyStatus, HouseholdSize, HouseholdCount

MATCH (person:RecordID)-[:HAS\_STATUS]->(status:PovertyStatus {status: PovertyStatus})

WITH PovertyStatus, HouseholdSize, HouseholdCount, COUNT(person) AS TotalCount

// Step 4: Calculate percentages for each household size in each PovertyStatus

RETURN

  PovertyStatus,

  HouseholdSize,

  ROUND((HouseholdCount \* 1.0 / TotalCount) \* 100, 2) AS Percentage

ORDER BY PovertyStatus, HouseholdSize;

**Is wealth evenly distributed across different regions of the United States?**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

// Step 1: Match RecordID nodes with their PovertyStatus and Location

MATCH (person:RecordID)-[:HAS\_STATUS]->(status:PovertyStatus)

MATCH (person)-[:LIVES\_IN]->(location:Location)

// Step 2: Count the number of people for each PovertyStatus and Location

WITH

  status.status AS PovertyStatus,

  location.state AS Location,

  COUNT(person) AS LocationCount

// Step 3: Calculate the total number of people within each PovertyStatus

WITH PovertyStatus, Location, LocationCount

MATCH (person:RecordID)-[:HAS\_STATUS]->(status:PovertyStatus {status: PovertyStatus})

WITH PovertyStatus, Location, LocationCount, COUNT(person) AS TotalInPovertyStatus

// Step 4: Calculate percentages for each Location within each PovertyStatus

RETURN

  PovertyStatus,

  Location,

  ROUND((LocationCount \* 1.0 / TotalInPovertyStatus) \* 100, 2) AS Percentage

ORDER BY PovertyStatus, Location;