Projecte Neo4j

[Repositori GitHub]

Exercici 1. Importeu les dades en la BD de Neo4j del projecte.

Tractament preeliminar de les dades:

Creació de restriccions (constraints)

Per garantir la integritat de les dades i evitar la creació de nodes duplicats o no identificables, s'han definit les següents restriccions:

```
// Clau primaria individu
CREATE CONSTRAINT individu_id_unique IF NOT EXISTS
FOR (i:Individu)
REQUIRE i.id IS UNIQUE;

// Clau primaria habitatges
CREATE CONSTRAINT habitatge_composite_key IF NOT EXISTS
FOR (h:Habitatge)
REQUIRE (h.id_llar, h.municipi, h.any_padro) IS NODE KEY;
```

```
neo4j$ CREATE CONSTRAINT individu_id_unique IF NOT EXISTS FOR (i:Individu) REQUIRE i.id IS UNIQUE
neo4j$ CREATE CONSTRAINT habitatge_composite_key IF NOT EXISTS FOR (h:Habitatge) REQUIRE (h.id_llar, h.mu…
```

Creació dels Index:

Amb l'objectiu de millorar el rendiment tant en la càrrega de dades com en l'execució de consultes, s'han creat diversos índexs:

Per al node habitatge:

```
// Index per any habitatge
CREATE INDEX habitatge_any_padro IF NOT EXISTS
FOR (h:Habitatge)
ON (h.any_padro);

// Index per municipi
CREATE INDEX habitatge_municipi IF NOT EXISTS
FOR (h:Habitatge)
ON (h.municipi);
```

```
// Index per direcció (carrer i número)
CREATE INDEX habitatge_adreca IF NOT EXISTS
FOR (h:Habitatge)
ON (h.carrer, h.numero);
```

```
neo4j$ CREATE INDEX habitatge_any_padro IF NOT EXISTS FOR (h:Habitatge) ON (h.any_padro)
neo4j$ CREATE INDEX habitatge_municipi IF NOT EXISTS FOR (h:Habitatge) ON (h.municipi)
neo4j$ CREATE INDEX habitatge_adreca IF NOT EXISTS FOR (h:Habitatge) ON (h.carrer, h.numero)
```

Per al node individu:

```
// Index per cognoms
CREATE INDEX individu_cognoms IF NOT EXISTS
FOR (i:Individu)
ON (i.cognom1, i.cognom2);
```

Per poder flexibilitzar i agilitzar les consultes que tinguin a veure amb el nom d'alguna persona també hem introduit el següent index

```
// Index de búsqueda full-text sobre nom y cognoms
CREATE FULLTEXT INDEX individu_fulltext_index IF NOT EXISTS
FOR (i:Individu)
ON EACH [i.nom, i.cognom1, i.cognom2];
```

```
neo4j$ CREATE INDEX individu_cognoms IF NOT EXISTS FOR (i:Individu) ON (i.cognom1, i.cognom2)
neo4j$ CREATE FULLTEXT INDEX individu_fulltext_index IF NOT EXISTS FOR (i:Individu) ON EACH [i.nom, i.cog…
SUCCESS  Added 1 index, completed after 247 ms.
```

Evitar valors erronis o nulls:

Tot i que les restriccions ja eviten valors nuls en les claus primàries, també s'aplica una lògica addicional per descartar valors com "null" o cadenes buides en tots els atributs:

```
SET node.atribut = CASE WHEN row.atribut IS NOT NULL AND row.atribut <> ' ' AND
toLower(row.atribut) <> 'null' THEN row.atribut ELSE NULL END
```

Aquesta estructura s'utilitza sistemàticament per assegurar la qualitat de les dades durant el procés de càrrega.

Conversions de tipus

Els fitxers CSV carregats interpreten tots els valors com a cadenes de text. Per millorar

l'eficiència, es realitzen conversions a tipus numèrics (enter) quan és pertinent.
Aquesta pràctica redueix el consum de memòria i accelera operacions com comparacions i ordenacions.

Els atributs convertits a enters inclouen: ID de l'individu, ID de l'habitatge, any del padró i número de carrer.

Carregar les dades:

El procés de càrrega es divideix en tres fases: creació de nodes, creació de relacions i neteja/validació de les dades. S'utilitzen instruccions `LOAD CSV` amb tractament de valors erronis i conversions de tipus.

```
// ====================
// 1. Crear nodes Individu
// ====================
LOAD CSV WITH HEADERS FROM 'file:///INDIVIDUAL.csv' AS row
WITH row
WHERE row.Id IS NOT NULL AND row.Id <> '' AND toLower(row.Id) <> 'null'
MERGE (i:Individu {id: toInteger(row.Id)})
SET i.nom = CASE WHEN row.name IS NOT NULL AND row.name <> '' AND
toLower(row.name) <> 'null' THEN row.name ELSE NULL END,
   i.cognom1 = CASE WHEN row.surname IS NOT NULL AND row.surname <> '' AND
toLower(row.surname) <> 'null' THEN row.surname ELSE NULL END,
   i.cognom2 = CASE WHEN row.second_surname IS NOT NULL AND
row.second_surname <> '' AND toLower(row.second_surname) <> 'null' THEN
row.second_surname ELSE NULL END,
   i.any_padro = CASE
      WHEN row.Year IS NOT NULL AND row.Year <> '' AND toLower(row.Year) <> 'null'
      THEN toInteger(row.Year)
      ELSE NULL
   END;
```

```
Added 17606 labels, created 17606 nodes, set 88030 properties, completed after 579 ms.
```

```
// ====================
// 2. Crear nodes Habitatge
// ====================
LOAD CSV WITH HEADERS FROM 'file:///HABITATGES.csv' AS row
WITH row
WHERE row.Id_Llar IS NOT NULL AND row.Id_Llar <> '' AND toLower(row.Id_Llar) <> 'null'
  AND row.Any_Padro IS NOT NULL AND row.Any_Padro <> '' AND
toLower(row.Any_Padro) <> 'null'
MERGE (h:Habitatge {
   id_llar: toInteger(row.Id_Llar),
   any_padro: toInteger(row.Any_Padro),
```

```
    municipi: row.Municipi
})
SET h.carrer = CASE WHEN row.Carrer IS NOT NULL AND row.Carrer <> '' AND
toLower(row.Carrer) <> 'null' THEN row.Carrer ELSE NULL END,
    h.numero = CASE
        WHEN row.Numero IS NOT NULL AND row.Numero <> '' AND toLower(row.Numero)
<> 'null'
        THEN toInteger(row.Numero)
        ELSE NULL
    END;
```

```
Added 3682 labels, created 3682 nodes, set 18410 properties, completed after 338 ms.
```

Una vegada que hem creat els nodes, afegim les relacions en forma d'arestes:

```
// ====================
// 3. Relació VIU (Individu)-[:VIU]->(Habitatge)
// ====================
LOAD CSV WITH HEADERS FROM 'file:///VIU.csv' AS row
WITH row
WHERE row.IND IS NOT NULL AND row.IND <> '' AND toLower(row.IND) <> 'null'
  AND row.HOUSE_ID IS NOT NULL AND row.HOUSE_ID <> '' AND
toLower(row.HOUSE_ID) <> 'null'
  AND row.Year IS NOT NULL AND row.Year <> '' AND toLower(row.Year) <> 'null'
MATCH (i:Individu {id: toInteger(row.IND)})
MATCH (h:Habitatge {id_llar: toInteger(row.HOUSE_ID)})
MERGE (i)-[:VIU {any_padro: toInteger(row.Year)}]->(h);
```

```
Set 68841 properties, created 68841 relationships, completed after 1305 ms.
```

```
// ====================
// 4. Relació SAME_AS (Individu)-[:SAME_AS]->(Individu)
// ====================
LOAD CSV WITH HEADERS FROM 'file:///SAME_AS.csv' AS row
WITH row
WHERE row.Id_A IS NOT NULL AND row.Id_A <> '' AND toLower(row.Id_A) <> 'null'
  AND row.Id_B IS NOT NULL AND row.Id_B <> '' AND toLower(row.Id_B) <> 'null'
MATCH (a:Individu {id: toInteger(row.Id_A)})
MATCH (b:Individu {id: toInteger(row.Id_B)})
MERGE (a)-[:SAME_AS]->(b);
```

```
Created 7199 relationships, completed after 390 ms.
```

```
// ====================
// 5. Relacions Familiars
// ====================
```

```
LOAD CSV WITH HEADERS FROM 'file:///FAMILIA.csv' AS row
WITH row
WHERE row.ID_1 IS NOT NULL AND row.ID_1 <> '' AND toLower(row.ID_1) <> 'null'
  AND row.ID_2 IS NOT NULL AND row.ID_2 <> '' AND toLower(row.ID_2) <> 'null'
  AND row.Relacio_Harmonitzada IS NOT NULL AND row.Relacio_Harmonitzada <> '' AND
toLower(row.Relacio_Harmonitzada) <> 'null'
MATCH (a:Individu {id: toInteger(row.ID_1)})
MATCH (b:Individu {id: toInteger(row.ID_2)})
MERGE (a)-[r:RELACIO]->(b)
SET r.tipus = row.Relacio_Harmonitzada;
```

Set 12022 properties, created 12022 relationships, completed after 675 ms.

Exercici 2. Resoleu les següents consultes Cypher:

Consulta a: Cognoms i habitants a Castellví de Rosanes per any de padró

Es recuperen tots els individus que han residit al municipi de *Castellví de Rosanes* (codificat com "CR"). Es combinen els primers i segons cognoms, es compten els habitants únics per any de padró, i finalment s'eliminen cognoms duplicats i el valor `"nan"`.

```
MATCH (h:Habitatge {municipi: "CR"})<-[:VIU]-(i:Individu)
WITH h.any_padro AS year,
    collect(i.cognom1) + collect(i.cognom2) AS cognoms_combinats,
    count(DISTINCT i) AS habitants
UNWIND cognoms_combinats AS cognom
WITH year, habitants,
    collect(DISTINCT cognom) AS cognoms_totals
WITH year, habitants,
    [c IN cognoms_totals
     WHERE c <> "nan"] AS cognoms
RETURN year, habitants, cognoms
ORDER BY year;
```

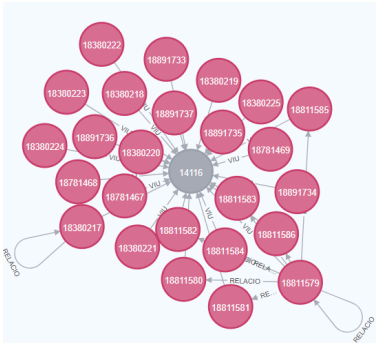
	year	habitants	cognoms
1	1866	1371	["galtes", "amigo", "corrons", "tort", "sanllehi", "galceran", "olle", "castells", "marti", "nin", "mas", "sala", "olive", "carcereny", "ribas", "suñol", "rusell", "planas", "sabat",

Consulta b: Trobada de veïns a partir d'un individu concret

Dada una persona identificada pel seu nom i primer cognom, es localitza l'habitatge on residia en un any i municipi concrets, i es recuperen tots els altres individus que vivien al mateix habitatge.

```
MATCH(r:Individu{nom:"rafel",cognom1:"marti"})-[v:VIU]->(h:Habitatg
```

```
e {any_padro: 1838, municipi: "SFLL"})<-[VIU]-(vei:Individu)
RETURN r, v, h, vei;
```



	Nom Vei	Cognoms Vei
1	"salvadora"	"nan nan"
2	"jpha"	"nan nan"
3	"jph"	"nan nan"
4	"maria"	"nan nan"
5	"felipe"	"nan nan"
6	"miquel"	"nan nan"
7	"franco"	"nan nan"

Consulta c: Variants d'identitat d'un individu (relacions SAME_AS)

S'identifica un individu a partir del seu nom complet i es recuperen totes les seves possibles variants mitjançant relacions SAME_AS, agrupant per nom i cognoms diferents.

```
MATCH(:Individu{nom:"miguel",cognom1:"estape",cognom2:"bofill"})-[:
SAME_AS]-(p)
RETURN collect(distinct p.nom) as `Variants Nom`, collect(distinct
p.cognom1) as `Variants Primer Cognom`, collect(distinct p.cognom2)
as `Variants Segon Cognom`
```

	Variants Nom	Variants Primer Cognom	Variants Segon Cognom
1	["miguel"]	["estape"]	["bofill", "bufill"]

Consulta d: Mitjana de fills per habitatge a Sant Feliu de Llobregat (1881)

Primer es calcula el número d'habitatges que hi habia a Sant Feliu de Llobregat l'any 1881 utilitzant un call. Després a partir de la gent que hi viu en aquell habitatge mirem els fills que viuen en aquell habitatge. Pot semblar redundant posar l'any de padró a la relació VIU, però es per evitar repeticions, ja que hi ha relacions viu que l'any de padró es diferent a l'any del padró de l'habitatge.

Després concretem la relació familiar a la relació filial, però que accepti tant a fills com a filles. Després contabilitzem el número de fills i fem la divisió.

```
CALL {
  MATCH (h:Habitatge {municipi: 'SFLL', any_padro: 1881})
  RETURN count(DISTINCT h) AS num_habitatges
}
WITH num_habitatges
```

```
MATCH (h:Habitatge {municipi: 'SFLL', any_padro: 1881})
MATCH(h)<-[:VIU{any_padro:1881}]-(:Individu)-[r:RELACIO]->(f:Indiv
idu)-[:VIU {any_padro: 1881}]->(h)
WHERE toLower(r.tipus) CONTAINS "fill"
WITH num_habitatges, count(DISTINCT f) AS total_fills
RETURN num_habitatges,total_fills,toFloat(total_fills)/num_habitat
ges AS mitjana_fills_per_habitatge
```

	num_habitatges	total_fills	mitjana_fills_per_habitatge
1	596	1299	2.1795302013422817

Consulta e: Famílies amb més de tres fills a Castellví de Rosanes

Primer es busca a tots els caps de familia de Castellví de Rosanes, eliminem els duplicats del same_as. Busquem els fills del cap de família, contem quants hi ha per cada cap i retornem la llista.

```
MATCH(i:Individu)-[:VIU]->(:Habitatge{municipi:"CR"}),(i)-[:RELACIO
]->(i)
where not exists{(i)<-[:SAME_AS]-(:Individu)}
match(i)-[r:RELACIO]->(f:Individu)
WHERE toLower(r.tipus) CONTAINS "fill"
with i, count(*) as fills
return "Familia "+i.cognom1+" " + i.cognom2 as Familia, fills
order by fills DESC
LIMIT 20
```

	Familia	fills
1	"Familia alsina vendrell"	10
2	"Familia astruch julia"	7
3	"Familia julivert parera"	6
4	"Familia bargallo ilegible"	6

Consulta f: Carrers amb menys habitants per any a Sant Feliu de Llobregat

El codi comença seleccionant totes les persones que viuen en habitatges del municipi SFLL i que tenen un carrer i un any del padró definits a continuació agrupa aquestes persones per any i carrer comptant el nombre d'habitants diferents que hi ha a cada carrer seguidament per a cada any es fa una subconsulta que busca el mínim nombre d'habitants que hi ha en algun carrer d'aquell mateix any i finalment es filtren només els carrers que tenen aquest

nombre mínim d'habitants per any després s'ordenen aquests carrers per any i per nom de carrer i es recull únicament el primer carrer de cada any per retornar finalment el carrer amb menys habitants i el seu nombre d'habitants ordenats de manera cronològica per any

```
MATCH (i:Individu)-[:VIU]->(h:Habitatge {municipi: "SFLL"})
WHERE h.carrer IS NOT NULL AND h.any_padro IS NOT NULL
WITH h.any_padro AS year, h.carrer AS carrer, count(DISTINCT i) AS
habitants
CALL {
  WITH year
  MATCH (i2:Individu)-[:VIU]->(h2:Habitatge {municipi: "SFLL"})
  WHERE h2.carrer IS NOT NULL AND h2.any_padro = year
  WITH h2.carrer AS carrer2, count(DISTINCT i2) AS habitants2
  RETURN min(habitants2) AS minHabitants
}
WITH year, carrer, habitants, minHabitants
WHERE habitants = minHabitants
WITH year, carrer, habitants
ORDER BY year, carrer
WITH year, head(collect({carrer: carrer, habitants: habitants})) AS
calleMin
RETURN year, calleMin.carrer AS carrer, calleMin.habitants AS
minHabitants
ORDER BY year
```

	year	carrer	minHabitants
1	1833	"carretera de la part de molins"	25
2	1838	"carretera de barna"	106
3	1839	"casas de camp del carrer de dal"	5
4	1878	"falguer"	13
5	1881	"s antonio"	22
6	1889	"Carretera"	16

Exercici 3. Analítica de Grafs:

En aquest exercici es realitza una anàlisi estructural del graf per tal d'obtenir una visió més profunda de la seva topologia i de les relacions entre els individus i els habitatges. Aquest tipus d'anàlisi és fonamental per detectar patrons de connexió, grups aïllats i possibles inconsistències en les dades.

---

a) Anàlisi de components connexes

Primer configurem els parametres necesaris per dur a terme aquest análisis, com el tipus de nodes i arestes a tenir en compte, el máxim de comunitats que pot detectar i el nom del graf projectat generat:

```
:param limit ⇒ ( 42);
:param config ⇒ ({});
:param communityNodeLimit ⇒ ( 1000);
:param graphConfig ⇒ ({nodeProjection: '*',relationshipProjection: {  relType: {    type: 'VIU',
orientation: 'UNDIRECTED',    properties: {}  }}});
:param generatedName ⇒ ('in-memory-graph-1749672849566');
```

```
neo4j$ :param limit ⇒ ( 42)
neo4j$ :param config ⇒ ({})
neo4j$ :param communityNodeLimit ⇒ ( 1000)
neo4j$ :param graphConfig ⇒ ({nodeProjection: '*',relationshipProjection: { relType: { type: 'VIU', orie…
neo4j$ :param generatedName ⇒ ('in-memory-graph-1749672849566')
```

En aquest cas utilitzem tots els nodes i les arestes només les de tipus VIU.

Projecció del graf:
El següent pas consisteix en fer la projecció del graf:

```
j$ CALL gds.graph.project($generatedName, $graphConfig.nodeProjection,
   $graphConfig.relationshipProjection, {});
```

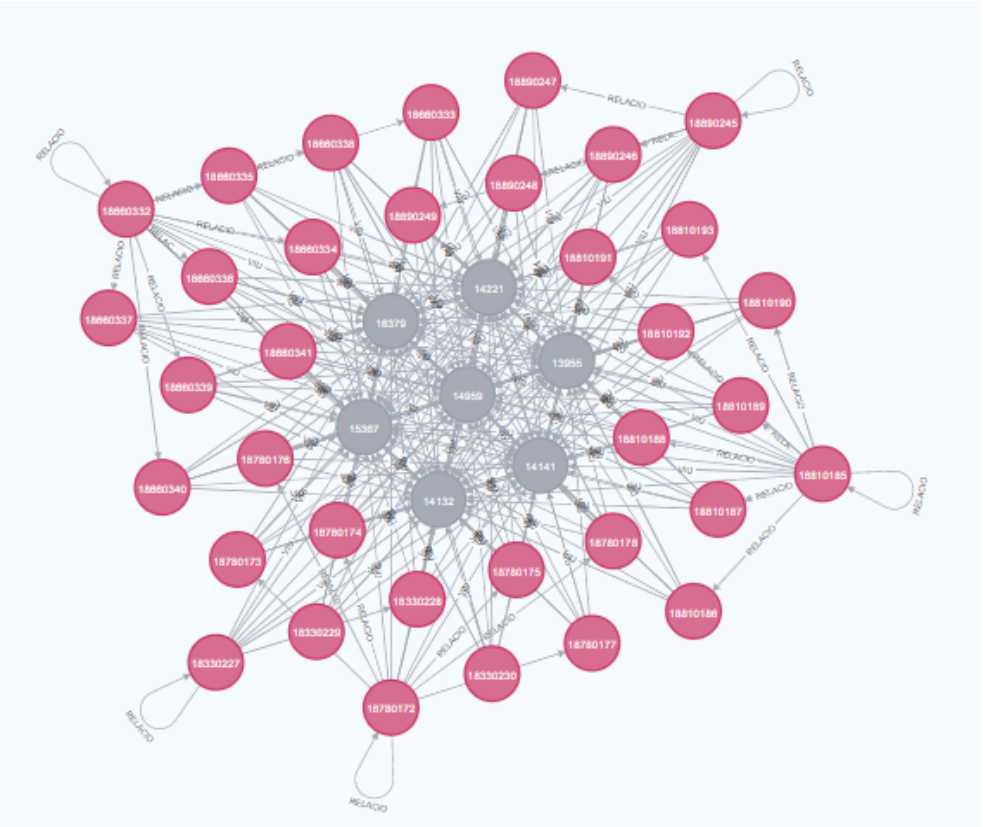
nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "__ALL__": { "label": "*", "properties": { } } }	{ "relType": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "VIU" } }	"in-memory-graph-1749672849566"	21288	137682	32

Execució de l'algorisme en mode *stream*:

```
CALL gds.wcc.stream($generatedName, $config) YIELD nodeId,
componentId AS community
WITH gds.util.asNode(nodeId) AS node, community
WITH collect(node) AS allNodes, community
RETURN community, allNodes[0..$communityNodeLimit] AS nodes,
size(allNodes) AS size
ORDER BY size DESC
LIMIT 1;
```

Aquest codi ens permet visualitzar la component més gran. La identificació de la component connex més gran en una base de dades de grafs permet centrar l'anàlisi en el subconjunt de dades amb més densitat relacional cosa que redueix el soroll i incrementa la qualitat dels resultats facilita l'aplicació d'algoritmes estructurals com la detecció de comunitats o la mesura de centralitat sobre un espai de dades consistent i interconnectat ajuda a detectar incoherències com individus duplicats o relacions improbables i proporciona una base fiable

per a inferències estructurals i validacions addicionals és una operació útil per estabilitzar el model abans de fer anàlisis més complexes i per assegurar que les conclusions es basen en la part més rellevant del conjunt de dades



a.1) Identificació de les components connexes més grans

S'obtenen les 10 components connexes amb més nodes. Aquesta informació és útil per detectar grups densament interconnectats, com ara nuclis familiars o conjunts d'habitatges relacionats.

Primer identifica a quina comunitat pertany cada node segons les connexions existents. Després recupera els nodes reals de cada comunitat i agrupa junts els que formen part de la mateixa. A continuació compta quants nodes té cada comunitat per mesurar-ne la mida. Finalment ordena les comunitats segons el nombre de nodes que contenen i retorna les deu més grans. Aquest procés permet saber quines són les parts més interconnectades del conjunt de dades.

```
CALL gds.wcc.stream($generatedName, $config)
YIELD nodeId, componentId AS community
WITH gds.util.asNode(nodeId) AS node, community
WITH community, collect(node) AS allNodes
RETURN community, size(allNodes) AS size
ORDER BY size DESC
LIMIT 10;
```

	community	size
1	5253	42
2	5158	40
3	5928	39
4	5179	38
5	5214	38
6	1872	37

Els resultats mostren que les 10 components connexes més grans tenen entre 37 i 42 nodes, la qual cosa indica una fragmentació clara del graf en grups relativament homogènis en mida això pot significar que no hi ha cap gran comunitat central dominant sinó diverses comunitats d'una mida similar que podrien representar famílies, grups de convivència o unitats socials amb connexions internes fortes La presència de components tan similars en nombre de nodes suggereix que la base de dades està formada per diversos grups socials relativament tancats i poc interconnectats entre si aquesta estructura podria indicar una població amb forta segmentació social o territorial on les relacions es concentren dins dels grups més petits i hi ha poca comunicació entre ells aquestes components poden reflectir l'existència de barreres socials o espacials que limiten la interacció entre diferents grups i podrien ser clau per identificar zones de dispersió o aïllament social La mida de les components també permet inferir que els estudis futurs haurien de centrar-se en aquestes unitats per entendre millor la dinàmica interna i com es mantenen aquestes comunitats aïllades o connectades amb la resta de la població

a.2) Quantitat de components sense cap node habitatge

Aquesta és una manera de comptabilitzar quanta gent no té cap habitatge.

```
CALL gds.wcc.stream($generatedName, $config)
YIELD nodeId, componentId AS community
WITH community, collect(gds.util.asNode(nodeId)) AS nodes
WITH community, [node IN nodes WHERE 'Habitatge' IN labels(node)]
AS habitatges
WHERE size(habitatges) = 0
RETURN count(community) AS components_sense_habitatge;
```

components_sense_habitatge
4741

El número del resultat no ens diu realment cuanta gent no té casa ja que hi ha duplicats, es a dir una persona que ha estat en 5 padrons sense casa, compte com 5 persones diferents.

b) Analisi de similaritat:

Primer creem l'aresta MATEIX_HAB:

```
MATCH (h1:Habitatge), (h2:Habitatge)
WHERE h1.municipi = h2.municipi
  AND h1.carrer = h2.carrer
  AND h1.numero = h2.numero
  AND h1.any_padro > h2.any_padro
MERGE (h1)-[:MATEIX_HAB]→(h2);
```

Table

Created 1752 relationships, completed after 395 ms.

Definim els parametres:

```
:param generatedName => 'in-memory-graph-1749677758442';
:param communityNodeLimit => 10;
:param limit => 42;
:param config => {
  similarityMetric: "Jaccard",
  similarityCutoff: 0.1,
  degreeCutoff: 1
};
```

Creem la projecció:

```
CALL gds.graph.project(
  $generatedName,
  '*',
  {
    VIU: {
      type: 'VIU',
      orientation: 'NATURAL'
    },
    RELACIO: {
      type: 'RELACIO',
      orientation: 'NATURAL'
    },
    MATEIX_HABITATGE: {
      type: 'MATEIX_HABITATGE',
      orientation: 'NATURAL'
    }
  }
);
```

Guardian els resultats de la similitud:

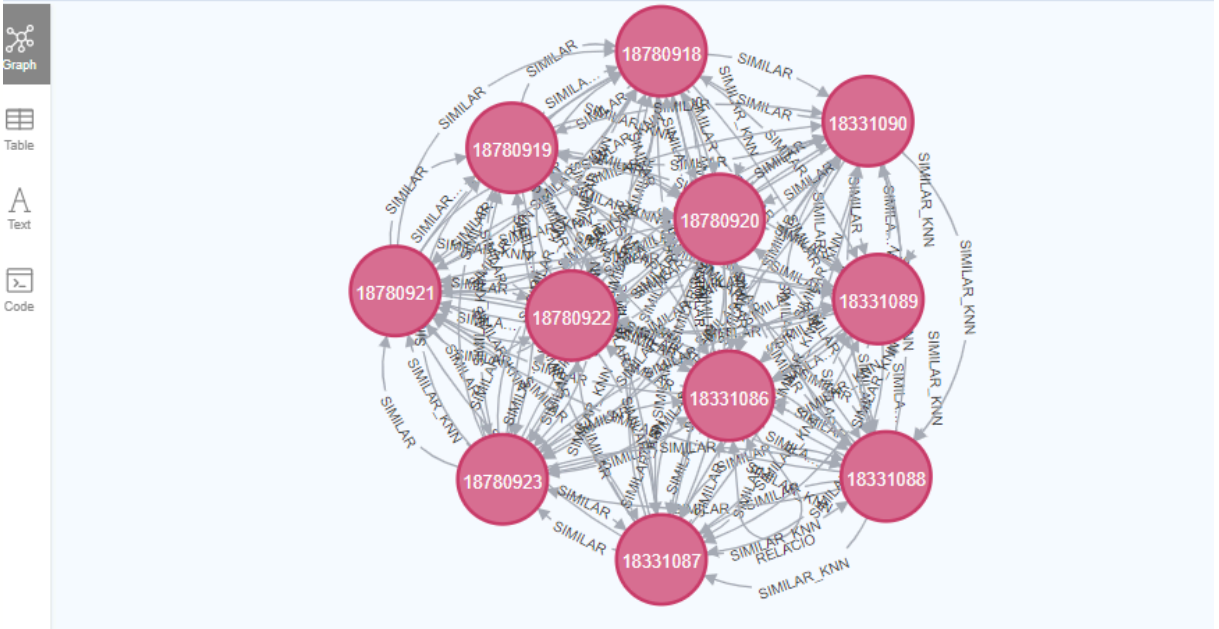
```
CALL gds.nodeSimilarity.write(
  $generatedName,
  {
    similarityMetric: 'Jaccard',
    similarityCutoff: 0.1,
    degreeCutoff: 1,
    writeRelationshipType: 'SIMILAR',
    writeProperty: 'similarity'
  }
)
YIELD nodesCompared, relationshipsWritten, similarityDistribution
RETURN nodesCompared, relationshipsWritten, similarityDistribution;
```

nodesCompared	relationshipsWritten	similarityDistribution
13373	126020	


```
{
  "min": 0.18181800842285156,
  "p5": 0.5454549789428711,
  "max": 1.000007629394531,
  "p99": 1.0000066757202148,
  "p1": 0.40000057220458984,
  "p10": 0.6666669845581055,
  "p90": 1.0000066757202148,
  "p50": 1.0000066757202148,
```


Exemple visualització similitud:

```
MATCH (a)-[r:SIMILAR]→(b)
RETURN a, r, b
LIMIT 50;
```



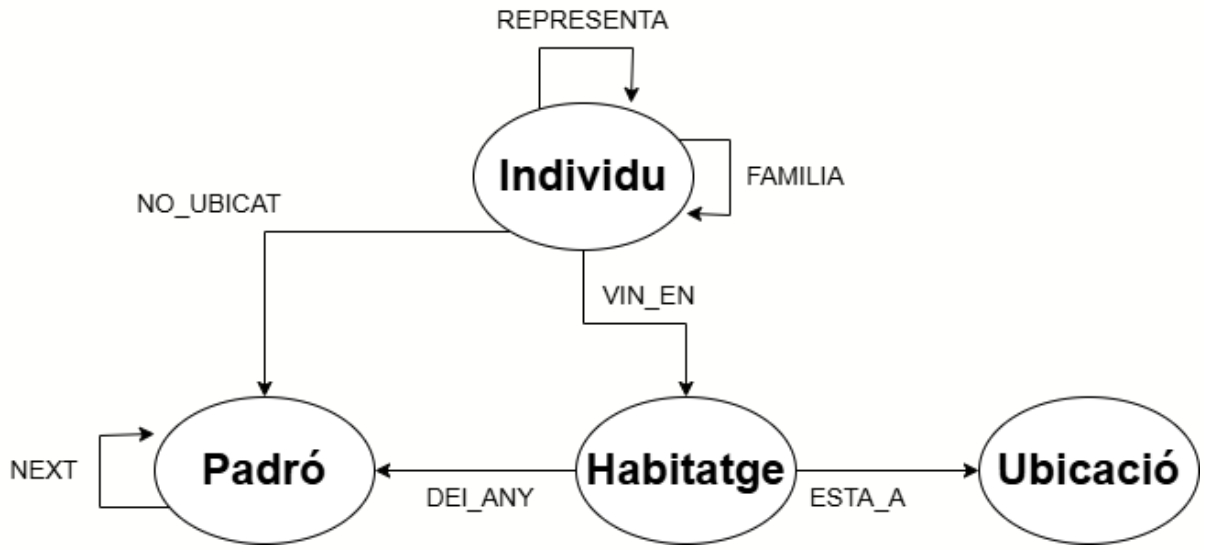
S'han comparat 13.373 nodes i s'han creat 126.020 relacions de similitud amb un llindar mínim de 0.1. La majoria de les similituds són molt altes, amb un valor mitjà de 0.94 i la major part dels percentils per sobre de 0.9, indicant que molts individus comparteixen connexions molt similars. Al cens, això vol dir que hi ha grups grans d'individus amb relacions molt semblants, com famílies o unitats de convivència gairebé idèntiques. Aquesta informació permet identificar agrupacions clares i possibles duplicats o errors, ajudant a ordenar i validar les dades de manera més precisa. També suggereix que és útil fixar-se en les parelles amb similitud més baixa per detectar casos amb relacions menys evidents o més complexes.

Exercici 4. Comparativa sobre diferents esquemes de base de dades:

Dissenyeu un altre esquema de la base de dades no relacional basada en grafs. En

concret: a) Dibuixeu l'esquema proposat. Ha de tenir com a mínim nodes de 3 tipus (és a dir, els labels no poden ser només de tipus habitatges i individus).

ESQUEMA PROPOSAT :



CANVIS REALITZATS:

1. Introducció del node Padró

Per tal de millorar l'estructura i la capacitat d'anàlisi temporal de la base de dades Neo4j, s'ha incorporat un nou node anomenat *Padró*. Aquest node representa explícitament els diferents anys en què s'ha realitzat el padró, facilitant així la navegació cronològica i l'estudi de l'evolució de les dades al llarg del temps.

El node *Padró* conté la propietat `any`, que indica l'any específic (per exemple, 1878, 1881, etc.). Per establir una seqüència temporal clara entre els diferents anys, s'ha creat la relació `NEXT`, que connecta cada node *Padró* amb el corresponent any següent.

2. Separació entre Habitatge i Ubicació

En el model original, el node *Habitatge* contenia l'atribut `location`, que descrivia l'adreça física directament. Aquesta aproximació, tot i ser funcional, limitava la capacitat del model per representar escenaris en què múltiples habitatges compartien la mateixa ubicació, o per analitzar les ubicacions de manera independent.

Amb la finalitat d'augmentar la claredat semàntica i la normalització del model, s'ha optat per separar el concepte d'*Habitatge* (un conjunt de persones convivint en un espai comú) del de *Ubicació* (l'adreça física concreta, incloent carrer, número i municipi).

3. Nova relació NO_UBICAT

Per tal de reflectir amb major precisió la realitat dels individus registrats, s'ha afegit la relació `NO_UBICAT` entre els nodes *Individu* i *Padró*. Aquesta relació identifica aquells individus que, per diferents motius, no poden ser associats a cap habitatge concret durant un any

determinat del padró.

4. Transició de cliques a un graf estel·lar

S'ha substituït la representació basada en cliques per un model de graf estel·lar, millorant així l'estructura i la facilitat d'interrogació de les dades.

SCRIPT D'IMPORTACIÓ

```
//Creamos restriccion de año y de individuo
CREATE CONSTRAINT padro_year_unique IF NOT EXISTS
FOR (p:Padro)
REQUIRE p.year IS UNIQUE;
CREATE CONSTRAINT individu_id_unique IF NOT EXISTS
FOR (i:Individu)
REQUIRE i.id IS UNIQUE;

//Creamos individuo y año
LOAD CSV WITH HEADERS FROM 'file:///INDIVIDUAL.csv' AS row
WITH row
WHERE row.Id IS NOT NULL AND row.Year IS NOT NULL
WITH row, ToInteger(row.Year) AS year
MERGE (p:Padro {year: year})
MERGE (i:Individu {id: row.Id})
SET i.nom = row.name,
    i.cognom = row.surname,
    i.segon_cognom = row.second_surname,
    i.year = year

//Convertimos año en una lista encadenada
MATCH (p:Padro)
WITH p ORDER BY p.year
WITH collect(p) AS years
UNWIND range(0, size(years)-2) AS i
WITH years[i] AS from, years[i+1] AS to
MERGE (from)-[:NEXT]->(to);

//Carreguem SAME_AS
LOAD CSV WITH HEADERS FROM 'file:///SAME_AS.csv' AS row
WITH row
WHERE row.Id_A IS NOT NULL AND row.Id_B IS NOT NULL

MATCH (a:Individu {id: row.Id_A})
MATCH (b:Individu {id: row.Id_B})
MERGE (a)-[:SAME_AS]->(b)



//Creem cliques relació
MATCH (n:Individu)
WHERE n.componentId IS NULL
CALL apoc.path.subgraphNodes(n, {relationshipFilter: 'SAME_AS'}) YIELD node
WITH n, collect(node) AS component
```

```
WITH component, id(n) AS cid
UNWIND component AS x
SET x.componentId = cid


//Creem nodes de referencia
MATCH (a:Individu)
WITH a.componentId AS group, a,
     size([x IN [a.Id, a.Year, a.name, a.surname, a.second_surname] WHERE x IS NOT
NULL]) AS filled
ORDER BY filled DESC
WITH group, collect(a)[0] AS rep

SET rep:Representant

//Creem la aresta referencia
MATCH (n:Individu)
MATCH (rep:Representant)
WHERE n.componentId = rep.componentId
MERGE (rep)-[:REPRESENTA]->(n)

//Borramos same as, componentId y la autoaresta
MATCH ()-[r:SAME_AS]-()
DELETE r;

MATCH (n)
REMOVE n.componentId

//importamos familia filtrando malos
LOAD CSV WITH HEADERS FROM 'file:///FAMILIA.csv' AS row
WITH row
WHERE row.ID_1 IS NOT NULL AND row.ID_2 IS NOT NULL
  AND row.Relacio_Harmonitzada IS NOT NULL
  AND NOT row.Relacio_Harmonitzada IN ['null', 'ala', 'al']

MATCH (a:Individu {id: row.ID_1})<-[:REPRESENTA]-(repA:Representant)
MATCH (b:Individu {id: row.ID_2})<-[:REPRESENTA]-(repB:Representant)

MERGE (repA)-[r:FAMILIA]->(repB)
SET r.relacio = row.Relacio_Harmonitzada

//importamos ubicación y habitatges y conexion entre ellos y con los años
LOAD CSV WITH HEADERS FROM 'file:///HABITATGES.csv' AS row
MERGE (u:Ubicacio {carrer: row.Carrer, numero: row.Numero, municipi: row.Municipi})
MERGE (h:Habitatge {id: row.Id_Llar, year: toInteger(row.Any_Padro), municipi:
row.Municipi})
MERGE (h)-[:ESTA_A]->(u)
WITH h, row

MATCH (a:Padro {year: toInteger(row.Any_Padro)})
MERGE (h)-[:DEL_ANY]->(a)

//Conectamos habitatge con los individuos
LOAD CSV WITH HEADERS FROM 'file:///VIU.csv' AS row
```

```
WITH row
MATCH (h:Habitatge{id:row.HOUSE_ID, year:toInteger(row.Year), municipi:
row.Location})
MATCH (:Individu {id: row.IND})<-[:REPRESENTA]-(repA:Representant)
MERGE (repA)-[:VIU_EN]->(h)

//Creamos arestas para individuos no conectados
MATCH (i:Individu)<-[:REPRESENTA]-(rep:Representant)
WHERE NOT EXISTS ((rep)-[:VIU_EN]->(:Habitatge))
WITH rep, i.year AS y
MERGE (p:Padro {year: y})
MERGE (rep)-[:NO_UBICAT]->(p)

//Creamos la segunda etiqueta de ubicacio
MATCH (u:Ubicacio)
CALL apoc.create.addLabels(u, [u.municipi]) YIELD node
REMOVE node.municipi

//Borramos los atributos redundantes
MATCH (h:Habitatge)
REMOVE h.municipi, h.year
```

CONSULTES CYPHER

Persones que sempre han viscut al mateix lloc

Model nou:

Es recuperen els representants que viuen en habitatges localitzats a una única ubicació, comprovant que el nombre d'ubicacions sigui exacte i que hagin residit en més d'una ocasió.

```
MATCH (rep:Representant)-[:VIU_EN]->(:Habitatge)-[:ESTA_A]->(u:Ubicacio)
WITH rep, size(collect(u)) AS n_ubicacions,  collect(DISTINCT u) AS ubicacions
WHERE size(ubicacions) = 1 AND n_ubicacions > 1
RETURN size(collect(rep)) AS SEDENTARIS
```

	SEDENTARIS
1	268

Model original:

Es crea un grup d'equivalència per individus amb relacions SAME_AS i es comprova que aquests grups tinguin una única adreça consistent.

```
MATCH (p:Person)

OPTIONAL MATCH (p)-[:SAME_AS]-(q:Person)
WITH p, collect(q) + p AS group_nodes
WITH [x IN group_nodes WHERE x IS NOT NULL | x.id] AS ids, group_nodes
WITH apoc.coll.sort(ids)[0] AS group_id, collect(DISTINCT p) AS persons
UNWIND persons AS person
MATCH (person)-[:VIU]->(h:Habitatge)
WITH group_id, collect(DISTINCT h.municipi + '|' + h.carrer + '|' + h.numero) AS adreces
WHERE size(adreces) = 1
RETURN count(*) AS SEDENTARIS
```

SEDENTARIS
6541

Persones que apareixen a dos padrons consecutius:

Model nou:

S'identifiquen els individus que viuen en habitatges associats a anys consecutius, utilitzant la relació `NEXT` per establir la continuïtat temporal.

```
MATCH (p1:Padro {year: $any_padro})-[:NEXT]->(p2:Padro)
MATCH (rep:Representant)-[:VIU_EN]->(:Habitatge)-[:DEL_ANY]->(p1)
WITH p1, p2, collect(DISTINCT rep) AS reps1
MATCH (rep:Representant)-[:VIU_EN]->(:Habitatge)-[:DEL_ANY]->(p2)
WITH reps1, collect(DISTINCT rep) AS reps2
WITH reps1, reps2,
    [r IN reps1 WHERE r IN reps2] AS enAmbdos
RETURN
  size(enAmbdos) AS persistents,
  size(reps1) AS primer_any,
  toFloat(size(enAmbdos)) / size(reps1) * 100 AS percentatge
```

persistents	primer_any	percentatge
1973	2744	71.90233236151603

Model original:

Es fa un seguiment més complex a través de les relacions `SAME_AS` per agrupar individus i verificar la seva presència en padrons consecutius.

```
MATCH (:Habitatge)
WHERE any_padro > year_actual
WITH year_actual, min(any_padro) AS year_seguent
```

```
MATCH (p:Person)-[:VIU]->(h1:Habitatge)
WHERE h1.any_padro = year_actual
WITH DISTINCT p, year_actual, year_seguent
OPTIONAL MATCH (p)-[:SAME_AS]-(q:Person)
WITH p, year_actual, year_seguent, collect(q.id) + p.id AS ids
WITH reduce(min_id = p.id, x IN ids | CASE WHEN x < min_id THEN x ELSE min_id END)
AS group_id, ids, year_actual, year_seguent
WITH group_id, ids, year_actual, year_seguent
WITH collect(DISTINCT group_id) AS grups_originals, year_actual, year_seguent
UNWIND grups_originals AS gid
MATCH (p:Person)-[:SAME_AS*0..]-(p_equiv:Person)
WHERE p.id = gid
WITH DISTINCT gid, collect(DISTINCT p_equiv.id) AS all_ids
UNWIND all_ids AS pid
MATCH (p2:Person {id: pid})-[:VIU]->(h2:Habitatge)
WHERE h2.any_padro = year_seguent
WITH DISTINCT gid AS grups_en_ambdós, count(DISTINCT gid) AS n_ambdós
WITH n_ambdós, count(*) + 0.0 AS total_grups
RETURN
  total_grups AS total_cens_inicial, n_ambdós AS persones_amb_cens_repetit,
  round(n_ambdós * 100.0 / total_grups, 2) AS percentatge;
```

total_cens_inicial	persones_amb_cens_repetit	percentatge
2744	1973	71.90233236151603

Després de comparar el codi Cypher per les mateixes consultes a la versió original i al nostre disseny, hem comprovat que **les consultes són molt més senzilles i clares** en el nostre model. Gràcies a una millor organització de les entitats i una normalització més acurada, **les relacions estan més definides i són més intuïtives**, fet que redueix la complexitat de les cerques i millora la llegibilitat i mantenibilitat del codi. Això facilita tant el desenvolupament com l'anàlisi de les dades.

AVANTATGES I DESAVANTATGES

D'una banda, la representació del temps mitjançant nodes Padró enllaçats cronològicament amb relacions NEXT permet una **reducció de la redundància i una millor normalització temporal**, facilitant el seguiment de l'evolució de les dades al llarg dels anys.

D'altra banda, passar de cliques a un graf estel·lar en les relacions familiars ha permès reduir considerablement el nombre d'arestes, cosa que implica un **estalvi de memòria** i una **eliminació de redundàncies familiars**, fent el model més eficient i net.

També s'ha separat el concepte d'Habitatge del d'Ubicació, permetent representar millor la diferència entre l'espai compartit per les persones i la seva localització física. Això comporta

una **millor organització del model i evita la generació de supernodes**.

Finalment, la incorporació de la relació NO_UBICAT entre individus i padrons dona cobertura als casos en què no es pot determinar el domicili d'una persona, fent possible la gestió de persones sense domicili conegut i oferint així un **model més complet i robust**.

És important destacar que, tot i els avantatges obtinguts amb la millora del model, també s'hi presenta un desavantatge: **l'augment de complexitat**. El fet de tenir un esquema més detallat i estructurat implica que els processos d'importació i modificació de la base de dades són més complexos, ja que cal gestionar més tipus de nodes, relacions i validacions. Aquesta complexitat pot traduir-se en un **major esforç de desenvolupament i manteniment**, especialment en entorns amb grans volums de dades o actualitzacions freqüents.

Participació:

Considerem que tots els membres del grup hem participat per al correcte desenvolupament del projecte.