Projecte de Neo4j: Padrons

Bases de Dades no Relacionals

Grau en Matemàtica Computacional i Analítica de Dades

Víctor Benito Segura (1597165)
Mireia Majó i Cornet (1597716)
Àlex Martín González (1605489)
Albert Roca Llevadot (1603375)

UAB
Universitat Autònoma de Barcelona

Índex

Introducció

En aquest projecte de Neo4j, treballarem amb un conjunt de dades que tracta sobre els padrons recopilats manualment cada 3-5 anys. D'aquest conjunt de dades podem extreure subconjunts que ens donaran informació sobre cada habitatge, les persones que hi viuen i la relació de parentesc que hi ha entre ells. És important mencionar que el canvi de base de dades relacional a base de dades no relacional (basada en grafs) se'ns proporciona a l'enunciat, per tant, no hem hagut de pensar l'estructura d'aquest.

Pel que fa al graf, podem veure que tenim dos tipus de nodes que es diferencien entre habitatges i individus que estan relacionats per viu, que ens representen el lloc on viu cada individu, la família, que és la relació de parentesc entre individus d'un mateix habitatge, i finalment, same_as que representa el mateix individu al llarg del temps.

Així doncs, l'objectiu principal d'aquest projecte és treballar, a partir d'un subconjunt de dades, la càrrega de dades, fer consultes a la base de dades i finalment, practicar l'ús d'algoritmes d'analítica de grafs.

Treball en equip

El projecte l'hem treballat tots conjuntament, procurant que cadascú assumís el mateix volum de feina. El primer lloc, presencialment vam entendre la pràctica i l'estructura que se'ns proposava de fitxer conjuntament amb els .csv on hi havia tota la informació. Tot seguit vam decidir com volíem fer el cypher i quin element que tenien NaNs eliminàvem i de quina manera els tractàvem. També vam debatre sobre les constrains que usaríem i els respectius índexs.

En segon lloc, un cop teníem clar com volíem fer tot el tractament de dades vam dur a terme diverses videotrucades on vam fer el codi de cypher tots junts, i finalment, un dels components actualitzava el programa al GitHub. En aquest codi vam haver de carregar totes les dades, generar els nodes, les relacions i finalment afegir les respectives característiques.

Tot seguit, un cop el codi va estar finalitzat i importàvem correctament la BD, ens vam repartir equitativament les consultes de la següent manera:

- Albert: 2, 6, 10
- Mireia: 3, 7, 11
- Àlex: 4, 8, 12
- Víctor: 5, 9, 13

És important remarcar que en les ocasions on un dels components del grup tenia una consulta més senzilla, ajudava als companys a finalitzar les seves.

En tercer lloc, ens vam tornar a reunir telemàticament per començar l'exercici 3, i vam acabar fent cadascú un apartat de la 3a. Vam fer les 3 primeres propostes i finalment vam afegir una més. Tot seguit, vam intentar fer el 3b.

Per acabar, ens hem repartit les seccions de l'informe final:

- Albert: Exercici 3 i Treball en equip
- Mireia: Exercici 2 i estructura del treball
- Àlex: Introducció i Exercici 1
- Víctor: Treball previ i README

Treball previ

En primer lloc, el que hem fet ha estat llegir l'enunciat de la pràctica per comprendre els objectius que se'ns proposaven.
Seguidament, hem observat els diferents arxius csv que contenien la informació de la base de dades. Cada arxiu estava format per les següents columnes:

Familia: [ID_1 , Relacio, Relacio_Harmonitzada, ID_2]

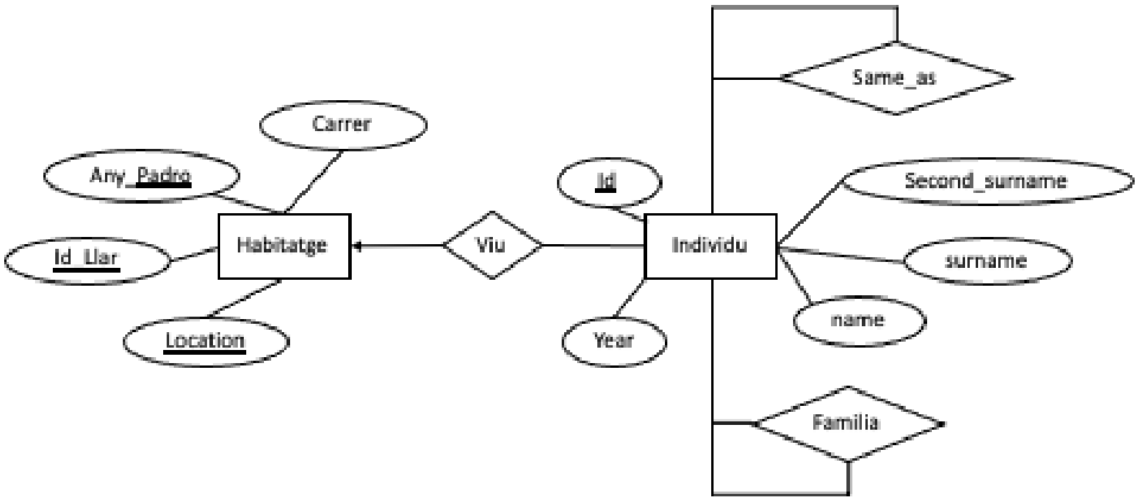
Habitatges: [Municipi, Id_Llar, Any_Padro, Carrer, Numero]

Individual: [Id, Year, name, surname, second_surname]

Same-as: [Id_A, SAME_AS, Id_B]

Viu: [IND , VIU, Location , Year, HOUSE_ID]

Aquestes dades estaven relacionades seguint el següent model Entitat-Relació:



Aquest model ha estat el nostre punt de partida. A partir d'aquí, crearem un cypher per migrar la informació a un sistema de gestió de bases de dades no relacionals com Neo4j on treballarem amb grafs.

Per fer tot aquest procés necessitarem prendre decisions en l'estructura que explicarem detalladament a l'exercici 1.

Creació del repositori de GitHub

Per concloure aquest apartat de treball previ, crearem un repositori a GitHub per tal d'anar actualitzant l'estat del projecte. En aquest repositori, realitzarem els commits de les diferents tasques que se'ns proposen:

Enllaç per accedir al repositori de GitHub

Exercicis

Exercici 1. Importa les dades en la BD de Neo4j del projecte. Genera un script en cypher que carregui totes les dades, generi tots els nodes, relacions i afegeixi les característiques allà on toqui.

Realitzarem una explicació del codi cypher per tal d'entendre què fa cada part del codi.

[1-2] Creació constraint pels nodes 'habitatge'

Definim una constraint (anomenada `habitatge_unic`) pel node habitatge per tal d'evitar crear nodes repetits. Definim els nodes únicament amb el municipi, l'identificador de la llar i l'any del padró (necessari ja que ens podem trobar el mateix habitatge en diferents anys).

[3-4] Creació constraint pels nodes 'individu'

En aquestes dues línies hem creat la constraint (anomenada `individu_unic`) pel node individu de tal manera que no hi hagi duplicions de nodes d'aquest tipus. En aquest cas ho mirarem a través de l'atribut `Id`, que és l'identificador de la persona i ha de ser únic.

[7-11] Creació nodes Individu

En aquestes línies el que fem és crear el node individu a partir del .csv "Individual". En aquest codi el que fem és llegir cada atribut i afegir-lo al node atribut, de tal manera que cada persona tingui les seves respectives característiques. És important remarcar que en aquest cas hem tractat els NaNs eliminant els registres que no tenien valor o bé en el nom o en el primer cognom (o NaN en els dos). Tot i això, hem afegit els valors que sí que tenien valor al nom i al primer cognom però no al segon.

[14 - 18] Creació nodes Habitatge

Aquest fragment de codi s'encarrega de tractar el .csv "Habitatges" per tal de poder crear el node Habitatge i els seus respectius atributs. En aquest cas descartarem els registres que no tinguin valors tant en municipi, com el número del carrer o de l'edifici, ja que vam considerar que si un d'aquests valors faltava era molt difícil fer una aproximació, perquè qualsevol d'aquests atributs pot fer variar molt la ubicació de l'individu.

[21-25] Creació relació familia

En aquest fragment de codi tractem el fitxer .csv "Familia" per tal de poder la relació entre individus per saber la seva relació de parentesc. El nom d'aquesta relació serà "familia" i l'atribut d'aquesta relació és el propi parentesc. És important tenir en compte que si la relació entre individus és nul·la hem decidit no descartar-la.

[28-32] Creació relació viu

En aquestes línies de codi hem tractat el fitxer .csv "Viu" que ens ha servit per crear la relació "viu" que relaciona el node habitatge amb el node individu. En aquest cas no ha fet falta fer cap tractament de NaNs. És important mencionar que els atributs de la relació serà el propi municipi, l'any del padró i finalment, l'identificador de la llar.

[35-39] Creació relació same_as

Per últim, hem tractat el .csv “same_as” que hem utilitzat per fer la relació same_as. Aquesta relació és entre nodes i individus i és la que ens permet representar la mateixa persona al llarg del temps.

Exercici 2. Resoleu les següents consultes:

1. **Del padró de 1866 de Castellví de Rosanes (CR), retorna el número d'habitants i la llista de cognoms, sense eliminar duplicats.**

```
MATCH (i:Individu)-[v:viu {Any_Padro:1866, Municipi:"CR"}]->(h:Habitatge)
WITH count(i) AS Recompte, collect(i.Cognom1) AS Cognom
RETURN Recompte, Cognom
```

2. **Per a cada padró de Sant Feliu de Llobregat (SFLL), retorna l’any de padró, el número d'habitants, i la llista de cognoms. Elimina duplicats i “nan”.**

```
MATCH (i:Individu)-[v:viu]->(h:Habitatge {Municipi:"SFLL"})
WITH count(i) AS Numero_Habitants, h AS h, collect(DISTINCT i.Cognom1) AS Cognoms
RETURN h.Any_Padro AS Any_Padro, Numero_Habitants, Cognoms
```

3. **Dels padrons de Sant Feliu de Llobregat (SFLL) d’entre 1800 i 1845 (no inclosos), retorna la població, l'any del padró i la llista d'identificadors dels habitatges de cada padró. Ordena els resultats per l'any de padró.**

```
MATCH (i:Individu)-[v:viu]->(h:Habitatge)
WHERE h.Municipi = "SFLL" AND i.Any_Padro > 1800 AND i.Any_Padro < 1845
RETURN COUNT(DISTINCT i) AS Poblacio, i.Any_Padro AS Any_Padro,
COLLECT(DISTINCT h.Id_Llar) AS Identificadors_Habitatges
ORDER BY i.Any_Padro
```

4. **Retorna el nom de les persones que vivien al mateix habitatge que "rafel marti" (no té segon cognom) segons el padró de 1838 de Sant Feliu de Llobregat (SFLL). Retorna la informació en mode graf i mode llista.**

```
MATCH (i:Individu {Nom:"rafel",
Cognom1:"marti"})-[:viu]->(h:Habitatge)<-[:viu]-(p:Individu)
WHERE h.Any_Padro = 1838 AND h.Municipi = "SFLL"
RETURN p.Nom AS Noms, i, h, p
```

5. **Retorna totes les aparicions de "miguel estape bofill". Fes servir la relació SAME_AS per poder retornar totes les instancies, independentment de si hi ha variacions lèxiques (ex. diferents formes d'escriure el seu nom/cognoms). Mostra la informació en forma de subgraf.**

```
MATCH (i:Individu)-[s:SAME_AS]-(r:Individu)
WHERE i.Nom = "miguel" AND i.Cognom1 = "estape" AND i.Cognom2 = "bofill"
RETURN i, s, r
```

6. **De la consulta anterior, retorna la informació en forma de taula: el nom, la llista de cognoms i la llista de segon cognom (elimina duplicats).**

```
MATCH (i:Individu)-[s:SAME_AS]-(r:Individu)
WHERE i.Nom = "miguel" AND i.Cognom1 = "estape" AND i.Cognom2 = "bofill"
RETURN i.Nom AS Nom, COLLECT(DISTINCT r.Cognom1) AS Cognom1,
COLLECT(DISTINCT r.Cognom2) AS Cognom2
```

7. **Mostra totes les persones relacionades amb "benito julivert". Mostra la informació en forma de taula: el nom, cognom1, cognom2, i tipus de relació.**

```
MATCH (i:Individu {Nom:"benito", Cognom1:"julivert"})-[f:familia]->(p:Individu)
RETURN p.Nom AS Nom, p.Cognom1 AS Cognom1, p.Cognom2 AS Cognom2,
f.Relacio AS Tipus_Relacio
```

8. **De la consulta anterior, mostra ara només els fills o filles de "benito julivert". Ordena els resultats alfabèticament per nom.**

```
MATCH (i:Individu {Nom:"benito", Cognom1:"julivert"})-[f:familia]->(p:Individu)
WHERE f.Relacio = "fill" OR f.Relacio = "filla"
RETURN p.Nom AS Nom, p.Cognom1 AS Cognom1, p.Cognom2 AS Cognom2,
f.Relacio AS Tipus_Relacio
ORDER BY p.Nom
```

9. **Llisteu totes les relacions familiars que hi ha.**

```
MATCH (i:Individu)-[f:familia]->(s:Individu)
RETURN i.Nom AS Individu_1, f.Relacio AS Relacio_Familiar, s.Nom AS Individu_2
```

10. **Identifiqueu els nodes que representen el mateix habitatge (carrer i numero) al llarg dels padrons de Sant Feliu del Llobregat (SFLL). Seleccioneu només els habitatges que tinguin totes dues informacions (carrer i numero). Per a cada habitatge, retorneu el carrer i número, el nombre total de padrons on apareix, el llistat d'anys dels padrons i el llistat de les Ids de les llars (eviteu duplicats). Ordeneu de més a menys segons el total de padrons i mostreu-ne els 15 primers.**

```
MATCH (h:Habitatge {Municipi:"SFLL"})
RETURN h.Carrer AS Carrer, h.Numero AS Numero, count(h) AS Nombre_Padrons,
COLLECT(h.Any_Padro) AS Anys_Padro
ORDER BY Nombre_Padrons DESC
```

11. **Mostreu les famílies de Castellví de Rosanes amb més de 3 fills. Mostreu el nom i cognoms del cap de família i el nombre de fills. Ordeneu-les pel nombre de fills fins a un límit de 20, de més a menys.**

```
MATCH (i:Individu)-[f:familia {Relacio:"jefe"}]->(i)
WHERE EXISTS((i)-[:viu]->(:Habitatge {Municipi:"CR"}))
WITH i, size([(i)-[:familia {Relacio:"fill"}]->() | 1])
+ size([(i)-[:familia {Relacio:"filla"}]->() | 1]) AS Nombre_Fills
WHERE Nombre_Fills > 3
RETURN i.Nom AS Nom, i.Cognom1 AS Cognom1, i.Cognom2 AS Cognom2, Nombre_Fills
ORDER BY Nombre_Fills DESC
LIMIT 20
```

12. Mitja de fills a Sant Feliu del Llobregat l'any 1881 per família. Mostreu el total de fills, el nombre d'habitatges i la mitja de fills per habitatge. Fes servir CALL per obtenir el nombre de llars.

```
MATCH (i:Individu)-[f:familia {Relacio:"jefe"}]->(:Individu)
MATCH (h:Habitatge {Municipi:"SFLL", Any_Padro: 1881})
WHERE EXISTS((i)-[:viu]->(h))
WITH i, size([(i)-[:familia {Relacio: 'fill'}]->() | 1]) + size([(i)-[:familia
{Relacio: 'filla'}]->() | 1]) AS num_fills, h.Id_Llar AS Id_Llar, h.Municipi AS
m, h.Any_Padro AS anyp
WITH COUNT(DISTINCT Id_Llar) AS num_habitatges, SUM(num_fills) AS total_fills
CALL {
    WITH total_fills, num_habitatges
    RETURN total_fills AS TotalFills, num_habitatges AS NumHabitatges,
toFloat(total_fills) / toFloat(num_habitatges) AS MitjanaFillsPerHabitatge
}
RETURN TotalFills, NumHabitatges, MitjanaFillsPerHabitatge
```

13. Per cada padró/any de Sant Feliu de Llobregat, mostra el carrer amb menys habitants i el nombre d'habitants en aquell carrer. Fes servir la funció min() i CALL per obtenir el nombre mínim d'habitants. Ordena els resultats per any de forma ascendent.

```
CALL {
  MATCH (i:Individu)-[r:viu]->(h:Habitatge)
  WHERE h.Municipi = "SFLL"
  WITH h.Any_Padro AS Any_Padro, h.Carrer AS Carrer, count(i) AS Num_Habitants
  ORDER BY Any_Padro ASC, Num_Habitants ASC
  RETURN Any_Padro, Carrer, Num_Habitants
}
RETURN Any_Padro, min(Carrer) AS Carrer, min(Num_Habitants) AS Min_Habitants
```

Exercici 3. En aquest exercici analitzarem les dades del graf per entendre millor l'estructura de les dades.

a) Estudi de les components connexes (cc) i de l'estructura de les component en funció de la seva mida.

```
CALL gds.graph.project(
  "graph2",
  ['Habitatge', 'Individu'],
  ['SAME_AS', 'familia', 'viu']
)
```

- **Taula agrupant els resultats segons la mida de la cc.**

```
CALL gds.wcc.stream('graph2')
YIELD componentId, nodeId
WITH componentId, collect(nodeId) AS nodes, size(collect(nodeId)) AS componentSize
ORDER BY componentSize ASC
RETURN componentSize, count(*) AS numberOfComponents
```

Amb aquestes línies de codi obtenim el següent output:

componentSize	numberOfComponents
1	585
2	106
3	48
4	37
5	16
6	17
7	10
8	4
9	3
10	2
22	1
24	1
14142	1

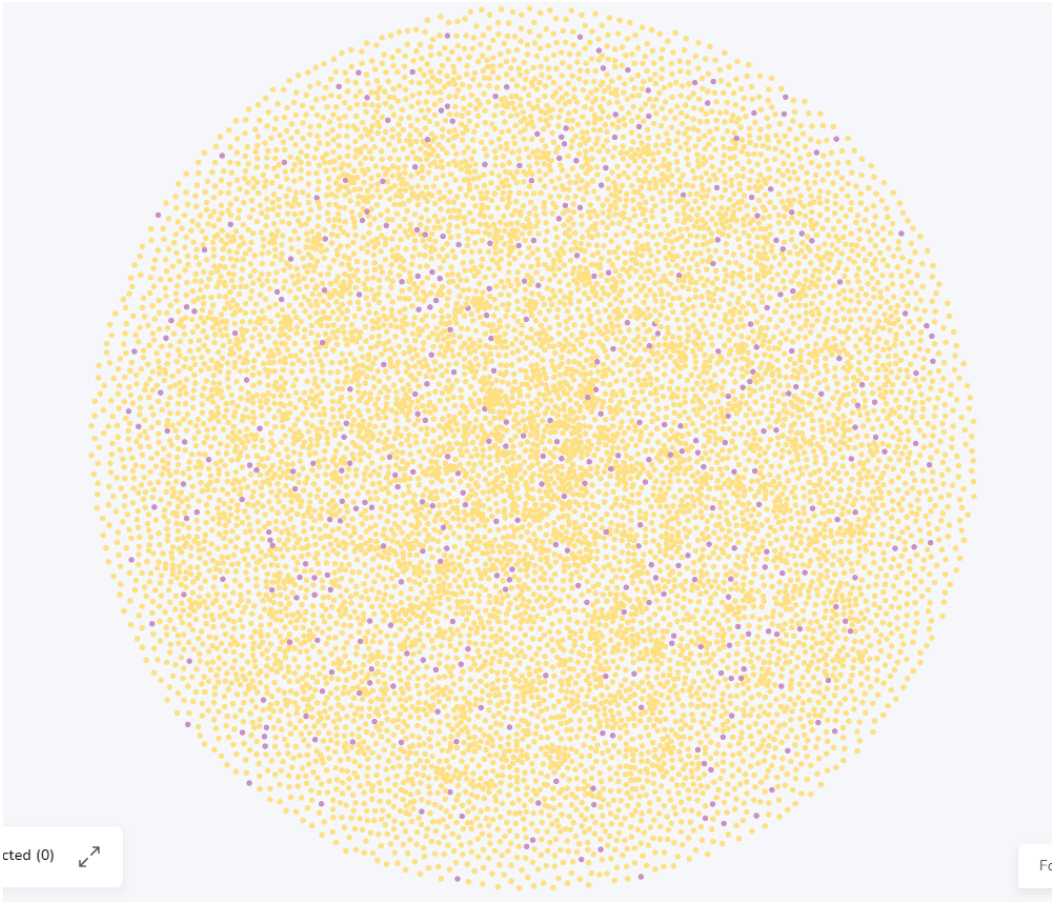
Veiem que ens dona que la component més gran és de mida 14142 i la segona més gran és de 24. Per comprovar-ho de manera visual utilitzarem el Bloom:

Component més gran:

```
CALL gds.wcc.stream('graph2')
YIELD componentId, nodeId
WITH componentId, collect(nodeId) AS nodes, size(collect(nodeId)) AS mida
ORDER BY mida DESC
LIMIT 2
WITH nodes
ORDER BY size(nodes) ASC
LIMIT 1
MATCH (n1)-[r]->(n2)
WHERE id(n1) IN nodes and id(n2) IN nodes
RETURN n1,r,n2;
```

D’on extraiem el primer graf:

Degut a la gran complexitat del graf veiem com no ens aconsegueix imprimir una gran part d’aquest, per tant, procedim a extreure les arestes i deixar únicament els nodes en el graf:

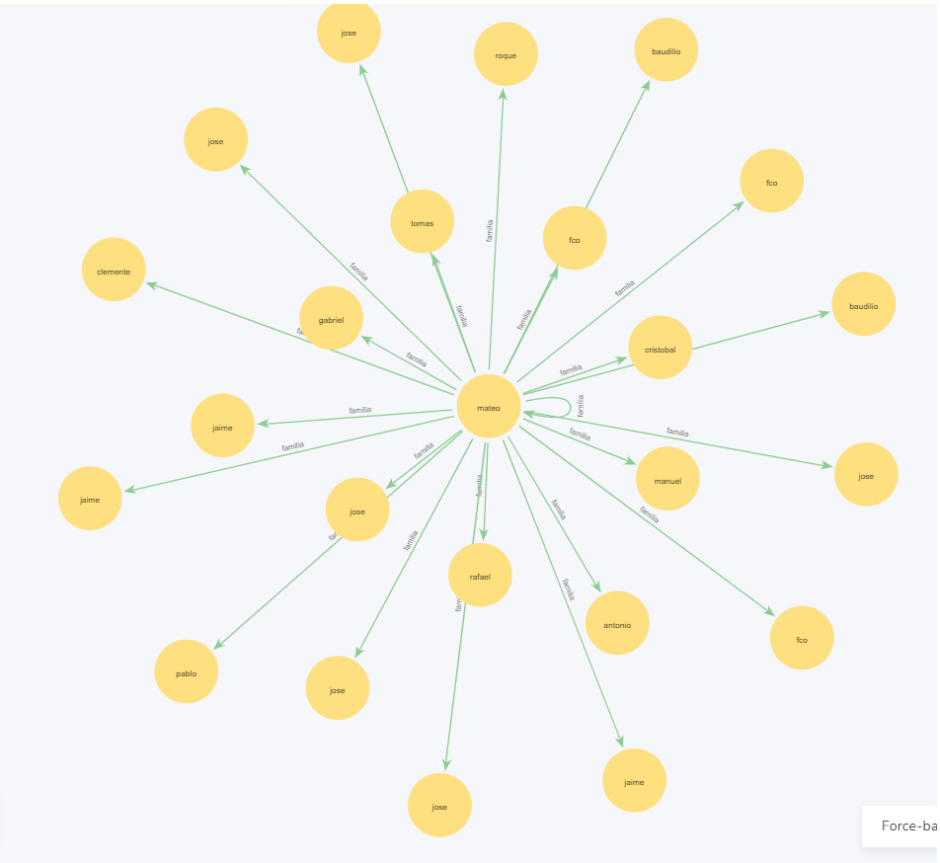


És important remarcar que els nodes grocs fan referència als individus i els liles a l’habitatge.

Segona component més gran:

```
CALL gds.wcc.stream('graph2')
YIELD componentId, nodeId
WITH componentId, collect(nodeId) AS nodes, size(collect(nodeId)) AS mida
ORDER BY mida DESC
LIMIT 2
WITH nodes
ORDER BY size(nodes) ASC
LIMIT 1
MATCH (n1)-[r]->(n2)
WHERE id(n1) IN nodes and id(n2) IN nodes
RETURN n1,r,n2;
```

D'on extraiem el següent graf:



En aquest observem el graf sencer, per tant, no fa falta treure cap aresta.

- **Distribució de tipus de nodes (Individu o Habitatge) segons la mida de la cc.**

```
CALL gds.wcc.stream('graph2')
YIELD componentId, nodeId
WITH componentId, collect(nodeId) AS nodes, size(collect(nodeId)) AS componentSize
MATCH (n)
WHERE id(n) IN nodes
WITH componentId, nodes, componentSize,
    CASE WHEN n:Individu THEN 1 ELSE 0 END AS individuCount,
    CASE WHEN n:Habitatge THEN 1 ELSE 0 END AS habitatgeCount
RETURN componentSize,
    sum(individuCount) AS n_Individu,
    sum(habitatgeCount) AS n_Habitatge
```

I com a sortida obtenim:

componentSize	n_Individu	n_Habitatge
14142	11753	2389
6	102	0
2	153	59
3	130	14
4	139	9
7	68	2
24	24	0
1	585	0
5	79	1
8	32	0
9	27	0
10	15	5
22	19	3

I en podem comprovar una part a les projeccions de Bloom anteriors, on podem veure que la component connexa de 24 nodes no té cap habitatge.

- **Per cada municipi i any el nombre de parelles del tipus: (Individu)—(Habitatge)**

```
match (:Individu)-[]->(h:Habitatge)
return h.Any_Padro,h.Municipi,count(*)
```

Obtenim el següent output:

h.Any_Padro	h.Municipi	count(*)
1866	"CR"	1067
1881	"SFLL"	9371
1878	"SFLL"	9052
1889	"SFLL"	9708
1838	"SFLL"	883
1833	"SFLL"	2485
1839	"SFLL"	4413

Per cada relació d'individus, mostrem la freqüència de cada tipus de relació per any

```
MATCH (i:Individu)-[r:familia]->(:Individu)
RETURN i.Year, apoc.coll.frequencies(collect(r.Relacio)) AS count
ORder by i.Year
```

Obtenim el següent output (en mostrem tan sols un fragment):

i.Year	count
1833	[{ "item": "jefe", "count": 310 }, { "item": "null", "count": 42 }]
1838	[{ "item": "jefe", "count": 51 }, { "item": "null", "count": 14 }]
1839	[{ "item": "null", "count": 118 }, { "item": "jefe", "count": 371 }]

b) Semblança entre els nodes. Ens interessa saber quins nodes són semblants com a pas previ a identificar els individus que són el mateix (i unirem amb una aresta de tipus SAME_AS).

```
MATCH (h:Habitatge)
WITH h.Carrer AS carrer, h.Municipi AS municipi, h.Numero AS numero, collect(h) AS
nodes, h.Any_Padro AS Year
WHERE size(nodes) > 1
WITH carrer, numero, municipi, min(Year) AS minYear
UNWIND nodes AS node
WITH node, carrer, numero, municipi, minYear
WHERE Year <> minYear
MATCH (minH:Habitatge {Any_Padro: minYear})
CREATE (node)-[:SAME_AS]->(minH)
```