**Projecte de Neo4J: Padrons**

# Treball en equip

En el següent informe es presenta un cas pràctic de l’ús de Neo4J. A continuació, s’exposa la repartició de tasques entre els components de l’equip:

1. **Tasca 1:** Aránzazu.
2. **Tasca 2:** Lucía.
3. **Tasca 3:** Jan i Cristian.
4. **Tasca 4:**
   1. Disseny BD: entre tots.
   2. Objectius i plantejament: Aránzazu.
   3. Creació BD (creació i modificació fitxers, importació dades): Lucía.
   4. Consultes (nou esquema i antic esquema): entre tots.
   5. Rendiment: Jan.
   6. Avantatges i inconvenients: Aránzazu.
   7. Power Point: entre tots.

L’evolució del treball es pot visualitzar al nostre GitHub:

*https://github.com/luciarevaliente/Neo4j\_1668160*

# 

# PART 1

## TASCA 1 - Importació de les dades en la BD de Neo4j

A continuació, mostrarem els passos seguits per la importació de dades a la base de dades Neo4j. Per fer-ho, hem utilitzats fitxers csv. A més, detallarem les relacions establertes entre els diferents nodes del graf creat. Per tal de mantenir la integritat i reduir la redundància, hem creat prèviament constraints i índexs, que també explicarem al llarg de l’apartat.

### CONSTRAINTS

Per tal de crear les constraints vam haver de pensar quines restriccions volíem implementar. A continuació, es mostren les constraints creades i la funció de cadascuna.

La primera constraint és **unique\_id\_person**. Ha estat creada per tal de que la Id de cada persona sigui únic i, per tant, no hi hagi 2 persones amb el mateix identificador. Això pot generar conflictes a l’hora de fer consultes ja que hi hauria molta redundància. La consulta que crea aquesta constraint és la següent:

CREATE CONSTRAINT unique\_id\_person FOR (i:INDIVIDUAL) REQUIRE i.Id is UNIQUE;

La segona constraint és **id\_personExists**. Aquesta constraint ha estat creada per tal de que la Id de cada persona no sigui NULL i per tant, existeixi. Això ens ajuda a mantenir la integritat de les dades i ens assegura que no es produiran errors a l’hora de fer consultes específiques d’algú. La comanda que crea aquesta constraint és la següent:

CREATE CONSTRAINT id\_personExists FOR (i:INDIVIDUAL) REQUIRE i.Id is NOT NULL;

La primera constraint és **id\_llarExists**. Aquesta constraint ha estat creada per tal de que la Id de cada llar no sigui NULL i per tant, existeixi. D’igual forma, això ens ajuda a controlar l’identificador de cada llar, mantenint la integritat de la BD. La consulta que crea aquesta constraint és la següent:

CREATE CONSTRAINT id\_llarExists FOR (h:HABITATGE) REQUIRE h.Id\_Llar is NOT NULL;

La Figura 1 mostra la sortida de Neo4j a l’executar la comanda SHOW CONSTRAINTS.

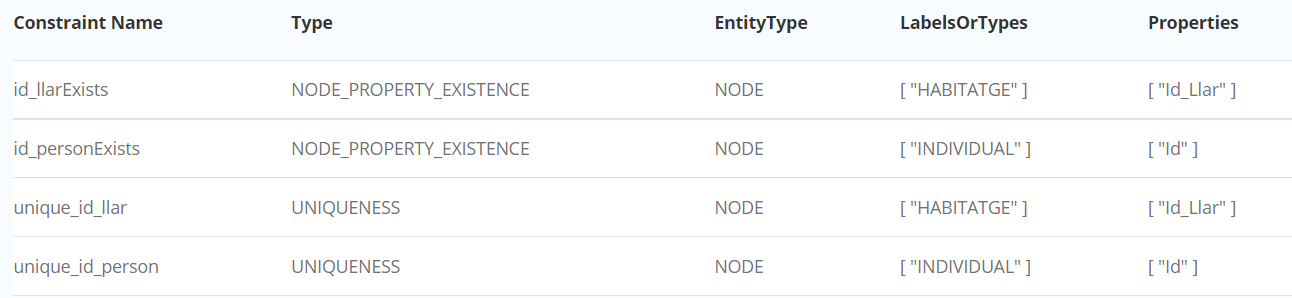


Figura 1

Per últim, vam deliberar si crear **unique\_id\_llar** era adient. D’aquesta manera, cada llar tindria Id única i, per tant, no hi hauria 2 llars amb la mateixa Id. Després d’observar i revisar els fitxers csv, ens vam adonar de que sí que podien existir dos habitatges registrats amb la mateixa Id. El motiu és perquè aquests registres es van fer en diferents anys. Per tant, si haguessim fet aquesta restricció, no podríem veure l’evolució d’un habitatge en el temps.

### ÍNDEXS

Per tal de crear els índexs vam haver d’analitzar les consultes que volíem realitzar. A continuació, es mostren els índexs creats i la funció de cadascun.

El primer índex és **AnyPadroIndex**. Aquest ha estat creat ja que volem fer moltes consultes filtrant l’any del padró. Per tant, com aquesta informació està continguda en una aresta i cercar en aquesta és estructura és ineficient, hem filtrar les arestes VIU millorant l’eficiència, tal i com veurem més endavant. La consulta que crea aquest índex és:

CREATE INDEX AnyPadroIndex FOR ()-[r:VIU]-() ON (r.Year)

El segon índex és **RelacioFamilia**. Aquest índex ha estat creat per tal de filtrar les arestes FAMILIA de manera més eficient per tal de trobar la relació de familia. El motiu és perquè fem moltes consultes on volem cercar els fills d’algú. D’igual forma que en el primer índex, és ineficient cercar informació continguda en arestes. La consulta que crea aquest índex és:

CREATE INDEX RelacioFamilia FOR ()-[f:FAMILIA]-() ON (f.Relacio\_Harmonitzada)

L’últim índex és **IndividuNomCognoms**. Aquest ha estat creat per tal d'identificar la informació dels individus al llarg del temps, més concretament amb els seus noms i cognoms. Hem decidit crear-ho ja que millorarà l’eficiència de les consultes, degut a que la majoria d’elles demanen filtrar els dades d’una persona en concret. La consulta que crea aquest índex és:

CREATE FULLTEXT INDEX IndividuNomCognoms FOR (i:INDIVIDUAL) ON EACH [i.name, i.surname, i.second\_surname]

La Figura 2 mostra la sortida de Neo4j a l’executar la comanda SHOW INDEXS.

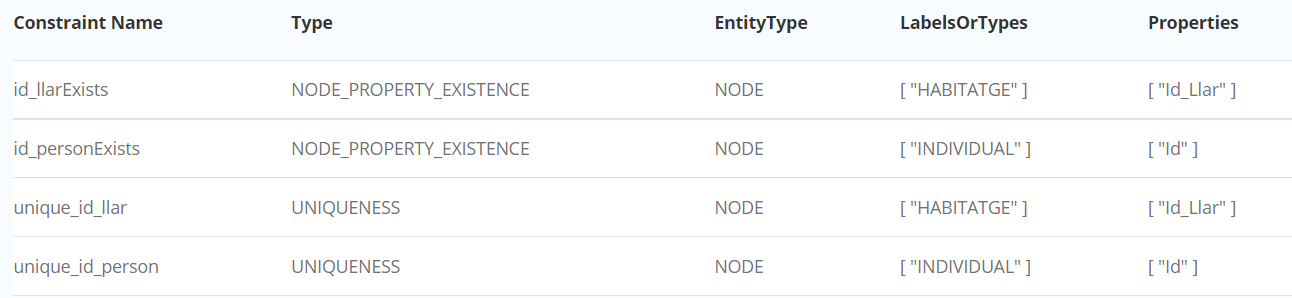


Figura 2

### IMPORTACIÓ DE DADES

Un cop creats els índexs i les constraints, vam dur a terme la importació de les dades a partir dels fitxers CSV mencionats.

En primer lloc, vam importar les dades dels habitatges. Per tal de crear els nodes HABITATGE, vam assegurar-nos que la Id\_Llar ni el Municipi fos NULL, mantenint la integritat de la BD. A continuació, vam inserir les propietats del node: Municipi, Id\_Llar, Any\_Padro, Carrer i Número. Per convertir les dades al seu tipus corresponent (int.) i filtrar que aquestes no fossin NULL, vam emprar la comanda CASE. La consulta de l’importació d’habitatges és la següent:

| LOAD CSV WITH HEADERS FROM 'file:/HABITATGES.csv' AS row  WITH row WHERE row.Id\_Llar <> "null" AND row.Municipi <> "null"  MERGE (h:HABITATGES { Municipi: row.Municipi, Id\_Llar: row.Id\_Llar, Any\_Padro: toInteger(row.Any\_Padro), Carrer: row.Carrer,Numero:  CASE WHEN row.Numero <> "null"  THEN toInteger(row.Numero)  ELSE row.Numero END }); |
| --- |

Figura 3

Seguidament, vam fer el mateix pels nodes de tipus INDIVIDUAL. Vam comprovar que l’Id no fos NULL, ja que era el que se’ns demanava. A més, éés necessari per a l’identificació de l’individu. A continuació, vam inserir inseríemles propietats del node: Id, Year (int), name, surname, second\_surname. La consulta de l’importació d’individus és la següent:

| LOAD CSV WITH HEADERS FROM 'file:\INDIVIDUAL.csv' AS row  WITH row WHERE row.Id <> "null"  MERGE (i:INDIVIDUAL { Id: row.Id, Year: toInteger(row.Year), name: row.name, surname: row.surname, second\_surname: row.second\_surname }); |
| --- |

Figura 4

Un cop inserides les dades dels nodes, vam inserir la informació de les arestes. Primer vam crear les relacions descrites a continuació:

Aquesta relació estableix que un individu (INDIVIDUAL) viu en un habitatge (HABITATGES). Els atributs de la relació són Location, que indica la ubicació, i Year, que especifica l'any en què es va registrar què l'individu vivia en aquest habitatge.

| LOAD CSV WITH HEADERS FROM 'file:///VIU.csv' AS row  WITH row WHERE row.IND <> "null" AND row.HOUSE\_ID <> "null"  MATCH (i:INDIVIDUAL), (h:HABITATGES)  WHERE row.IND = i.Id AND row.HOUSE\_ID = h.Id\_Llar  CREATE (i)-[:VIU {Location:row.Location,Year: toInteger(row.Year)}]->(h); |
| --- |

Figura 5

La següent relació identifica que dos individus (INDIVIDUAL) són la mateixa persona. Aquesta relació s'utilitza per consolidar registres duplicats o per indicar equivalències entre registres.

| LOAD CSV WITH HEADERS FROM 'file:///SAME\_AS.csv' AS row  WITH row WHERE row.Id\_A <> "null" AND row.Id\_B <> "null"  MATCH (a:INDIVIDUAL),(b:INDIVIDUAL)  WHERE row.Id\_A = a.Id AND row.Id\_B = b.Id  CREATE (a)-[:SAME\_AS]->(b); |
| --- |

Figura 6

En últim lloc, aquesta relació descriu el vincle familiar entre dos individus (INDIVIDUAL). Els atributs de la relació són Relacio, que especifica el tipus de relació familiar, i Relacio\_Harmonitzada.

| LOAD CSV WITH HEADERS FROM 'file:///FAMILIA.csv' AS row  WITH row WHERE row.ID\_1 <> "null" AND row.ID\_2 <> "null"  MATCH (a:INDIVIDUAL), (b:INDIVIDUAL)  WHERE row.ID\_1 = a.Id AND row.ID\_2 = b.Id  CREATE (a)-[:FAMILIA {Relacio: row.Relacio, Relacio\_Harmonitzada: row.Relacio\_Harmonitzada}]->(b); |
| --- |

Figura 7

## TASCA 2

### Consulta A)

La primera consulta demana: “per a cada padró (any) de Sant Feliu de Llobregat (SFLL), retorna l’any de padró, el nombre d'habitants, i la llista de cognoms. Elimina duplicats i “nan”.”

La consulta Cypher que hem realitzat té dues parts. En la primera cerquem els nodes INDIVIDUAL que han viscut a Sant Feliu de Llobregat. Per tant, cerquem el padró mitjançant l’aresta VIU. Tal com hem explicat en la tasca anterior, hem creat un Índex per la propietat Year de l’aresta VIU. Aquest s’utilitza automàticament en emprar “v.Year IS NOT NULL”, ja que no hi ha cap altre índex basat en aquest mateix atribut.

Per cada any, la consulta guarda: l’any del padró, el nombre d’habitants i una llista amb els cognoms dels habitants. Amb els distincts del WITH, aconseguim que per cada llista no es repeteixi cap cognom. Tot i això, quan les sumem, hem de tornar a filtrar-les perquè potser una persona es diu Joan Garcia Garcia. Per poder fer aquest segon filtratge, hem hagut d’emprar l’unwind (aquesta comanda ens permet agrupar els cognoms). Finalment, hem fet un altre distinct al return per eliminar duplicats de la suma.

A continuació, mostrem el resultat de la primera part de la consulta:

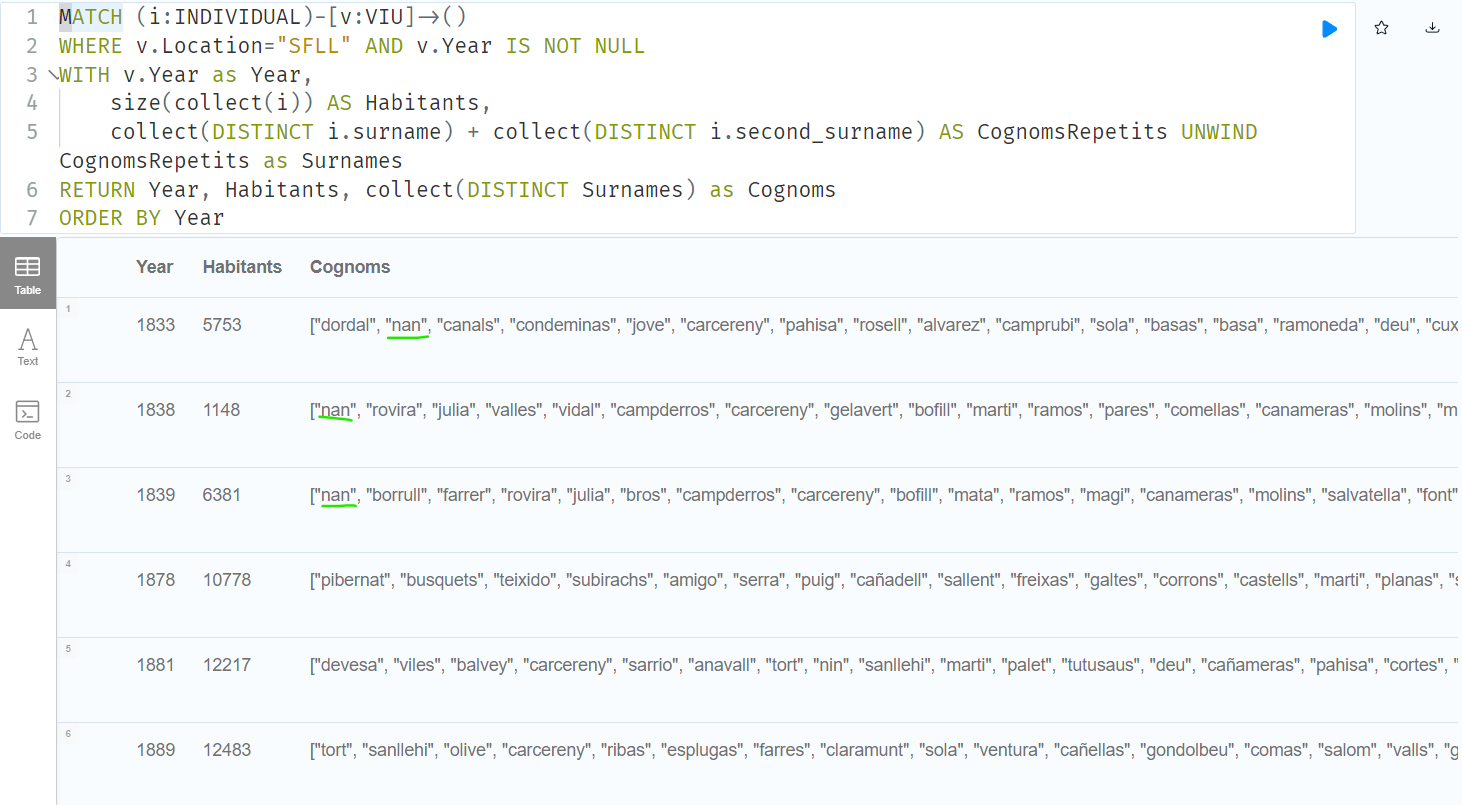


Figura 8

Tal com podem observar, les dades contenen valors “nan”. Per tant, hem de netejar-les. Per fer-ho, hem introduït aquest primer MATCH en un CALL, és a dir, en una subconsulta. D’aquesta manera, dividim el problema en parts més petites per aïllar possibles errors de sintaxi, entre d’altres.

Hem decidit fer l’eliminació dels cognoms “nan” després de la subconsulta i no en el primer WHERE, ja que si no eliminàvem anys del padró. Això té sentit perquè si es detectava un sol cognom amb aquest atribut, es descartava Year sencer.

La query final és la que es mostra a continuació. Tal com hem explicat, la segona part s’encarrega d’eliminar els valors invàlids gràcies a una list comprehension.

CALL{

MATCH (i:INDIVIDUAL)-[v:VIU]->()

WHERE v.Location="SFLL" AND v.Year IS NOT NULL

WITH v.Year as Year,

size(collect(i)) AS Habitants,

collect(DISTINCT i.surname) + collect(DISTINCT i.second\_surname) AS CognomsRepetits UNWIND CognomsRepetits as Surnames

RETURN Year, Habitants, collect(DISTINCT Surnames) as Cognoms

ORDER BY Year

}

WITH Year, Habitants, [cognom in Cognoms WHERE cognom <> "nan"] as CognomsFiltrats

RETURN Year, Habitants, CognomsFiltrats

El resultat de la consulta és la següent taula, on hi ha les columnes “Year”, “Habitants” i “CognomsFiltrats”:



Figura 9

### CONSULTA B)

La segona consultademana: “retorna totes les aparicions de "miguel estape bofill". Fes servir la relació SAME\_AS per poder retornar totes les instàncies, independentment de si hi ha variacions lèxiques (ex. diferents formes d'escriure el seu nom/cognoms). Mostra la informació en forma de taula: el nom, la llista de cognoms i la llista de segon cognom (elimina duplicats)”.

La consulta Cypher que hem realitzat es mostra a continuació. Aquesta consta de dues parts. La primera és una subconsulta continguda al CALL. Mitjançant l’índex de text creat en la tasca anterior, busquem el node més semblant a les propietats que cerquem. En altres paraules, obtenim el node que té aquest nom i cognoms. Per fer-ho, no ordenem els resultats de semblança per l’score, ja que el resultat de l’índex ja retorna els nodes ordenats per l’score de forma descendent. És per això que només retornem el primer resultat.

A continuació, fem una cerca dels nodes més similars al node trobat. Per tant, busquem les arestes amb label SAME\_AS. Dels nodes connexos, fem una llista dels noms amb els respectius cognoms.

La consulta Cypher és:

CALL{

CALL db.index.fulltext.queryNodes('IndividuNomCognoms', 'miguel estape bofill')

YIELD node, score

RETURN node

LIMIT 1

}

MATCH (i:INDIVIDUAL)-[:SAME\_AS]->(i2)

WHERE i.name=node.name AND i.surname=node.surname AND i.second\_surname=node.second\_surname

RETURN i.name as Nom, collect(DISTINCT i2.surname) as PrimerCognom, collect(DISTINCT i2.second\_surname) as SegonCognom

El resultat de la consulta és la següent taula.



Figura 10

Com podem observar, només hi ha una fila. Això significa que el nom del senyor Miguel Estape Bofill no ha patit modificacions ortogràfiques al llarg dels anys.

### CONSULTA C)

La tercera consulta demana: “mostra els fills o filles (només) de "benito julivert". Mostra la informació en forma de taula: el nom, cognom1, cognom2, i tipus de relació. Ordena els resultats alfabèticament per nom”.

La consulta Cypher que hem realitzat es mostra a continuació. Consta d’un primer filtratge idèntic a la consulta de l’apartat anterior. En aquest s’utilitza l’índex “IndividuNomCognoms” per cercar a “benito julivert” i retornar el node corresponent.

A partir d’aquí, es fa un MATCH per obtenir els seus fills. En aquest segon filtratge, especifiquem que només ens interessen les relacions “filla” i “fill”. Només ens guardem inforEls resultats els ordenem mitjançant el nom dels descendents, per ordre alfabètic.

CALL{

CALL db.index.fulltext.queryNodes('IndividuNomCognoms', 'benito julivert')

YIELD node, score

RETURN node

LIMIT 1

}

MATCH (n:INDIVIDUAL)<-[f:FAMILIA]-(m:INDIVIDUAL)

WHERE n = node AND f.Relacio\_Harmonitzada IN ["fill", "filla"]

WITH m.name as Nom, m.surname as Cognom, m.second\_surname as SegonCognom, f.Relacio\_Harmonitzada as Relacio

RETURN Nom, Cognom, SegonCognom, Relacio

ORDER BY Nom

El resultat de la consulta és la següent taula:

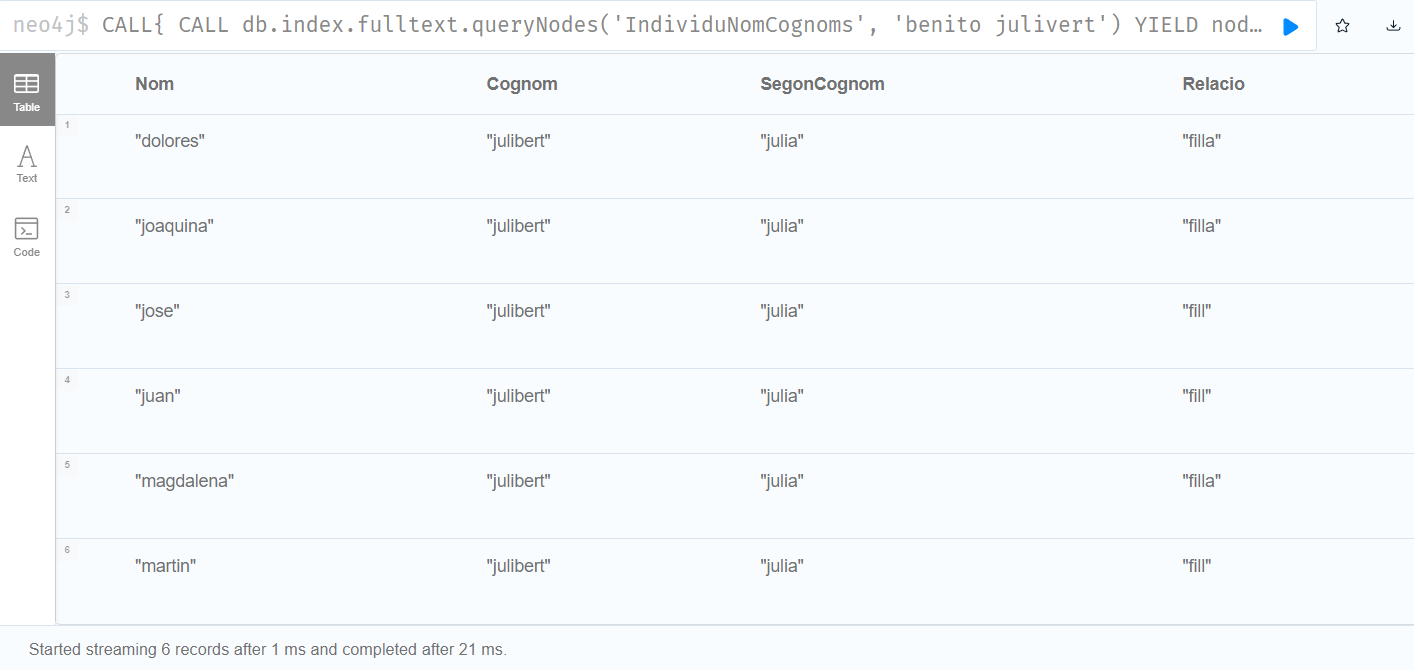


Figura 11

Es pot observar com en “benito julivert” en tenia sis fills, tots amb la mateixa dona. La meitat dels descendents van ser homes i l’altra meitat, dones.

### CONSULTA D)

La quarta consulta demana: “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”.

La consulta Cypher filtra les persones que viuen a Castellví de Rosanes. A més, s’assegura que aquestes tinguin fills o filles. Després de filtrar, només guardem la informació del cap de família i de quants fills en té. Més tard, imposem que el nombre de fills ha de ser major a 3. Retornem el nom, cognom, segon cognom i nombre de fills del cap de família. Finalment, ordenem pel nombre de fills descendentment fins a un límit de 20.

MATCH (:HABITATGES)<-[v:VIU {Location:"CR"}]-(i:INDIVIDUAL) <-[f:FAMILIA]-(:INDIVIDUAL)

WHERE f.Relacio\_Harmonitzada IN ["fill", "filla"]

WITH i.name as Nom, i.surname as Cognom, i.second\_surname as SegonCognom, count(f) as numFills

WHERE numFills>3

RETURN Nom, Cognom, SegonCognom, numFills

ORDER BY numFills DESC

LIMIT 20

El resultat de la consulta és la següent taula:

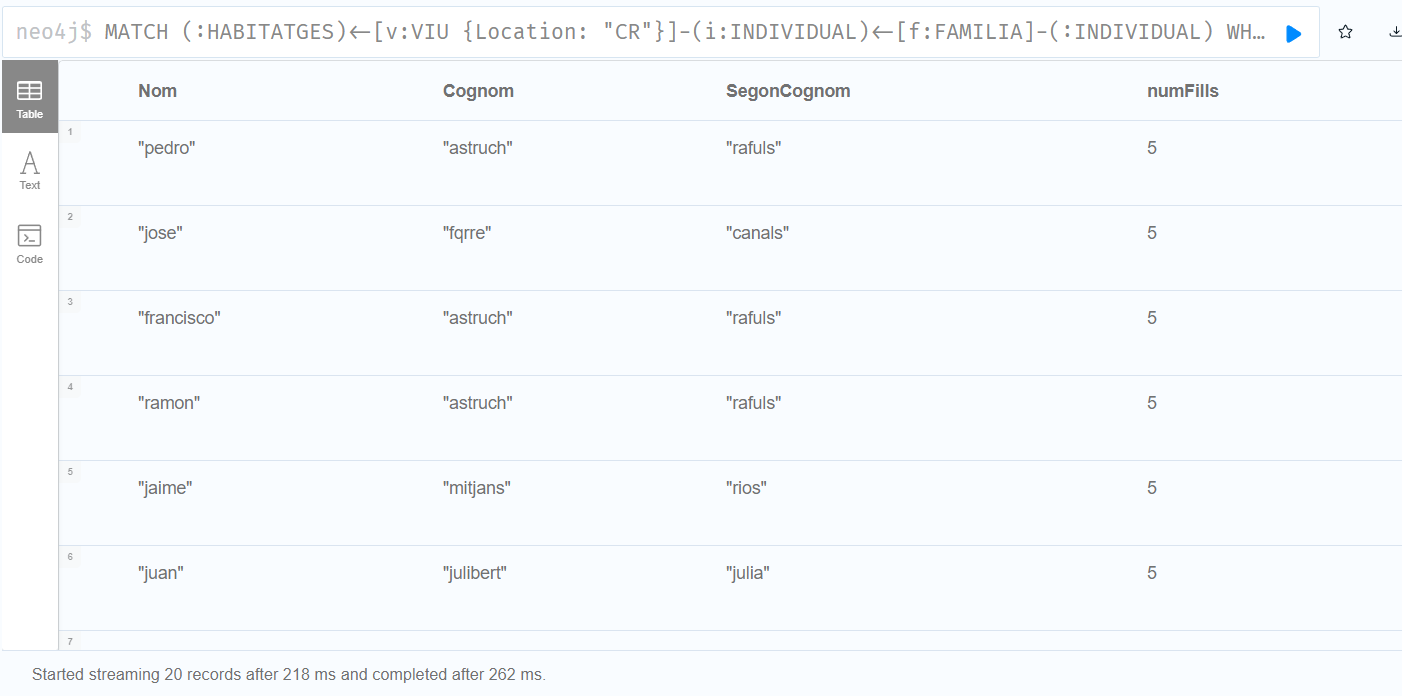


Figura 12

Observem que no hi ha cap persona de Castellví de Rosanes que tingui més de 5 fills.

Per acabar, volem destacar que per cercar els fills dels caps de família hem utilitzat la propietat Relacio\_Harmonitzada. Tot i això, hem fet la cerca també amb la propietat Relacio i obtenim els mateixos resultats. Per tant, no hi ha cap diferència entre ambdues ja que els resultats en totes dues consultes són 153 persones amb més de 3 fills.

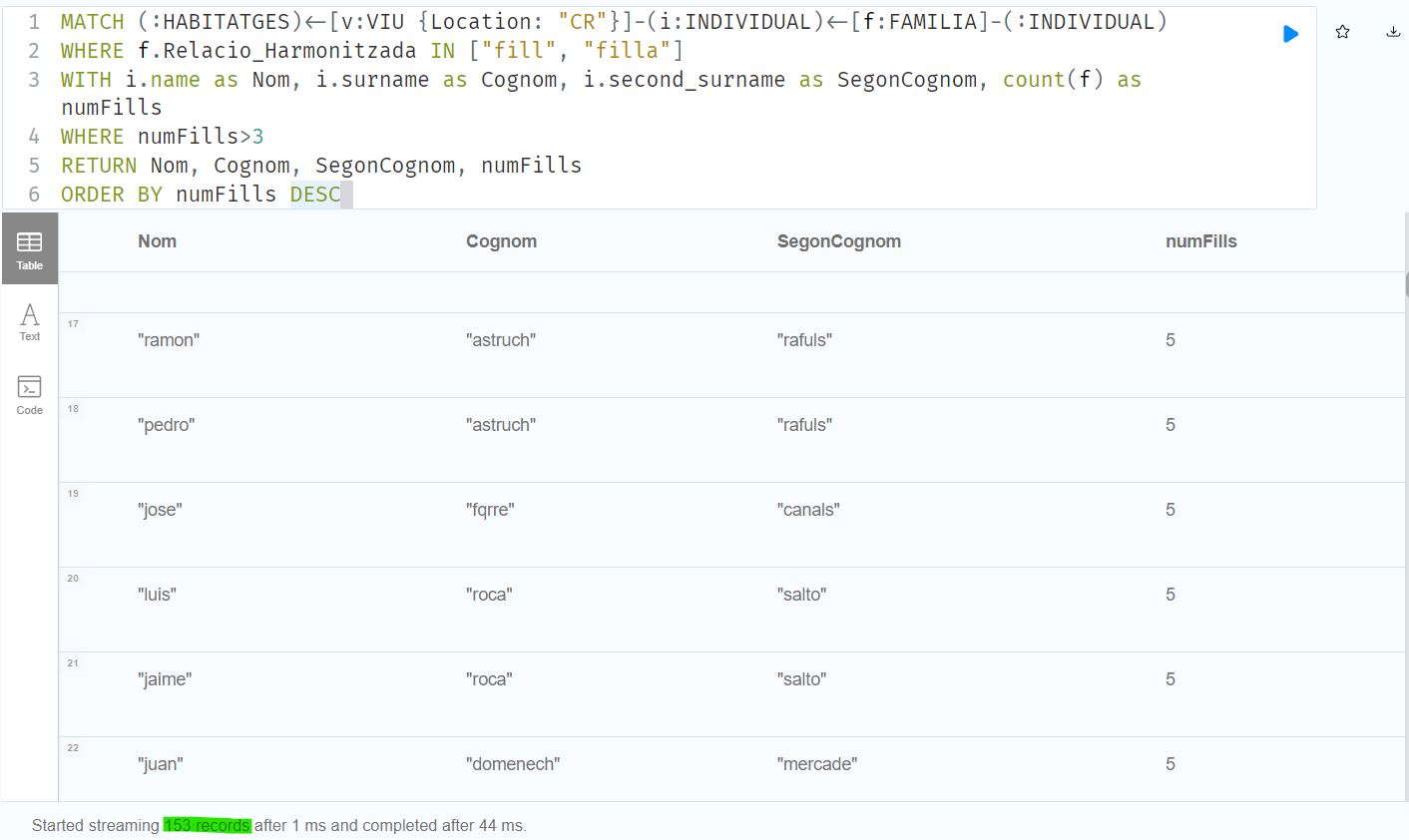


Figura 13

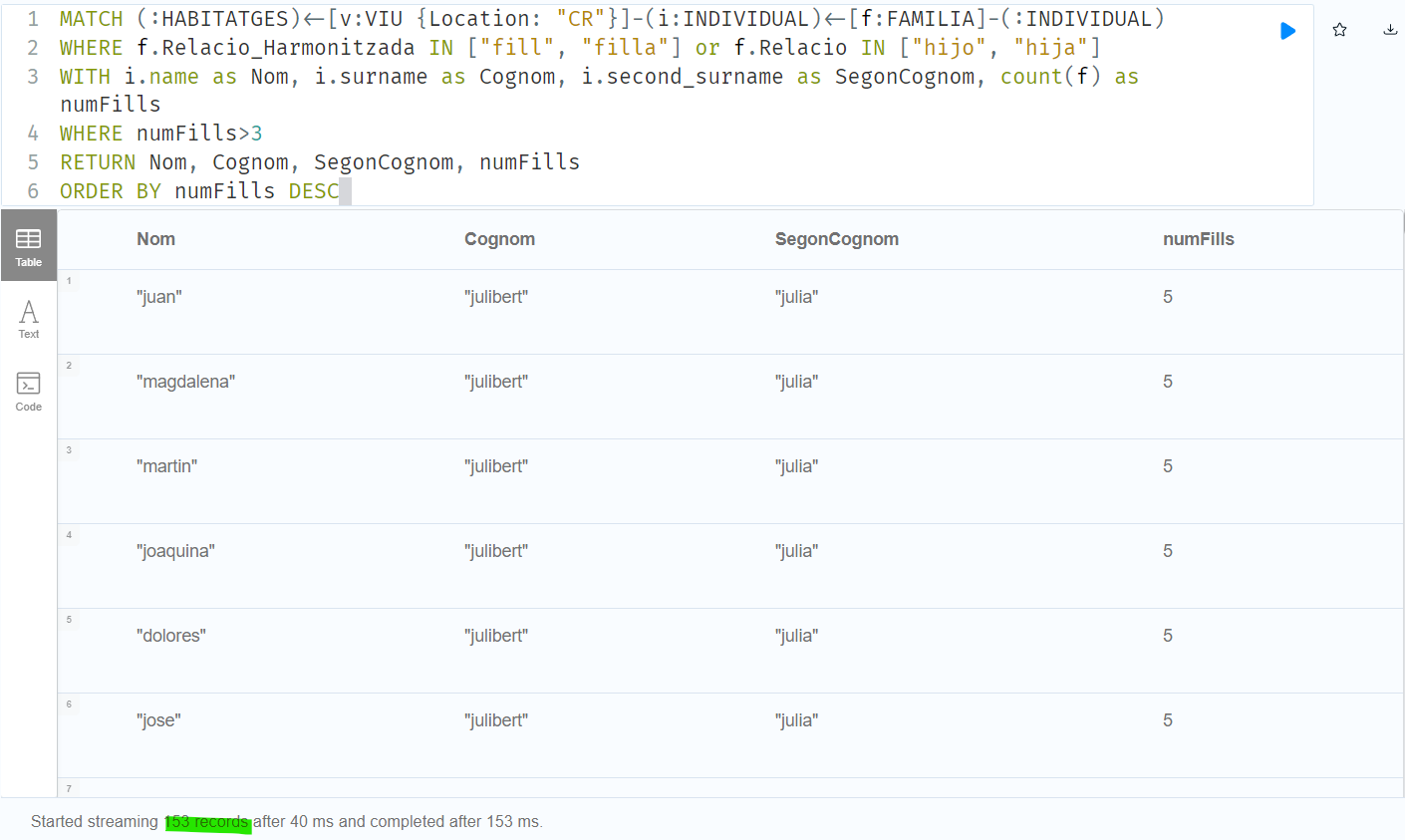


Figura 14

### CONSULTA E)

L'última consulta demana: “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”.

Tal com demana l’enunciat, el Cypher consta de dues parts. La primera s’encarrega de buscar els individus que viuen a Sant Feliu de Llobregat. De tota la informació, només guardem els anys, els carrers i el nombre d’habitants. Finalment, fem servir la funció min() per retornar només el mínim nombre d’habitants per cada padró, tal com es mostra a continuació.

MATCH (i:INDIVIDUAL)-[v:VIU]->(h:HABITATGES)

WHERE v.Location="SFLL"

WITH v.Year as Any\_Padro, h.Carrer as Carrer, count(i) as Habitants

RETURN Any\_Padro, min(Habitants) as minHabitants

La subconsulta anterior, l’emmagatzemem en un CALL. D’altra banda, fem un MATCH per tornar a trobar els individus que viuen a Sant Feliu de Llobregat i comptar els habitants per carrer i any. Guardem aquesta informació, juntament amb el valor mínim trobat i el seu any corresponent. Tot i que en les querys anteriors no hem guardat els valors retornats al CALL, en aquesta és necessari ja que utilitzem el WITH. Per tant, si no escrivim les variables minHabitants i Any\_Padro, s’eliminaran i no es podran utilitzar al WHERE. A l’acabar, retornem, per cada any, els carrers que tenen un nombre d’habitants equivalent al mínim. Finalment, ordenem els resultats per padró.

CALL{

MATCH (i:INDIVIDUAL)-[v:VIU]->(h:HABITATGES)

WHERE v.Location="SFLL"

WITH v.Year as Any\_Padro, h.Carrer as Carrer, count(i) as Habitants

RETURN Any\_Padro, min(Habitants) as minHabitants

}

MATCH (i:INDIVIDUAL)-[v:VIU]->(h:HABITATGES)

WHERE v.Location="SFLL"

WITH v.Year as Padro, h.Carrer as Carrer, count(i) as Habitants, minHabitants, Any\_Padro

WHERE Habitants=minHabitants and Padro=Any\_Padro

RETURN Any\_Padro, Carrer, Habitants

ORDER BY Any\_Padro

El resultat de la consulta és la següent taula:

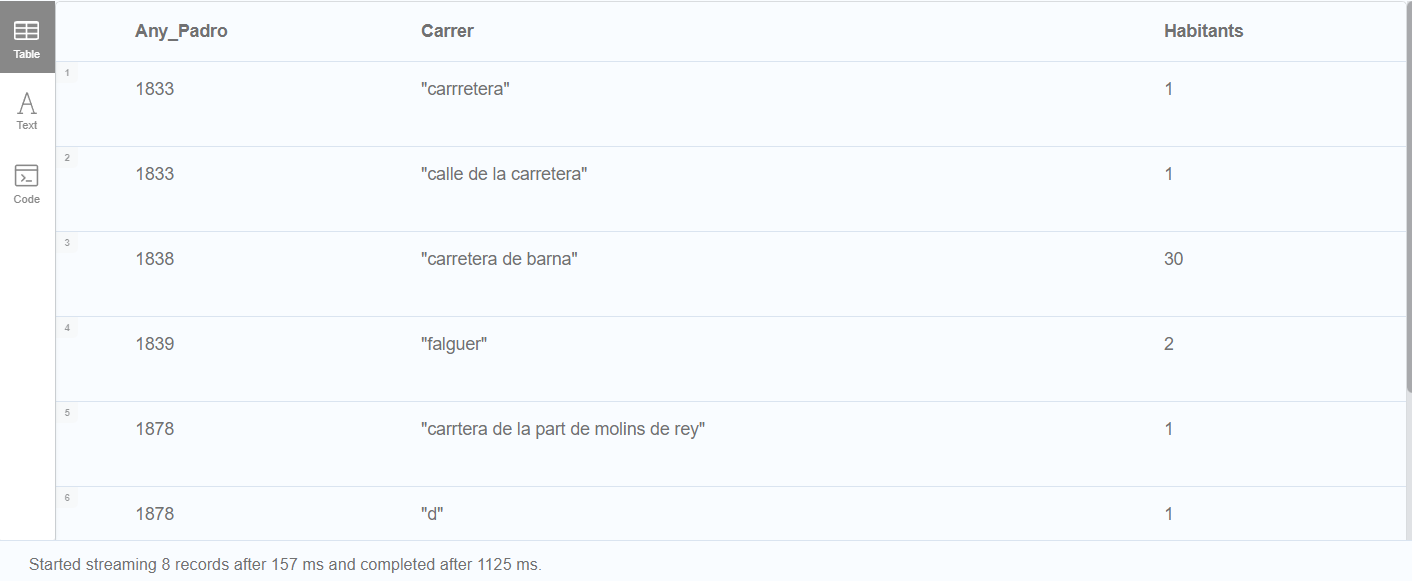


Figura 15

Observem que hi ha més files que anys. En total hi ha 6 padrons, per tant, per cada any hi ha més d’un carrer amb el mínim nombre d’habitants. Per exemple, l’any 1833 hi ha dos carrers amb 1 habitant. D’altra banda, el carrer amb un nombre mínim d’habitants més gran és el 1838. Podem concloure que va ser l’any en què més població hi havia a Sant Feliu de Llobregat.

## 

## 

## ANÀLISI RENDIMENT ÍNDEXS

Fent les consultes, ens hem adonat que si es refactoritzés el graf, es podrien millorar el rendiment de les consultes. Tal com hem vist a les classes de teoria, és més ràpid cercar labels que no pas propietats de nodes o arestes. És per aquest motiu que hem creat els índexs de la Tasca 1. A continuació, mostrarem el temps d’execució de les consultes amb índex i sense índex. Cal destacar que les execucions s’han repetit diverses vegades des del mateix ordinador, corroborant així les acceleracions mostrades i reduint els possibles errors.

Si executem la consulta A) sense índex, observem com el temps d’execució és de 235ms. En canvi, si ho fem amb l’índex AnyPadroIndex, aquest decreix als 189ms. Per tant, obtenim una acceleració del . Finalment, concloem que l’índex fa que la consulta s’executi 1.25 vegades més ràpid.

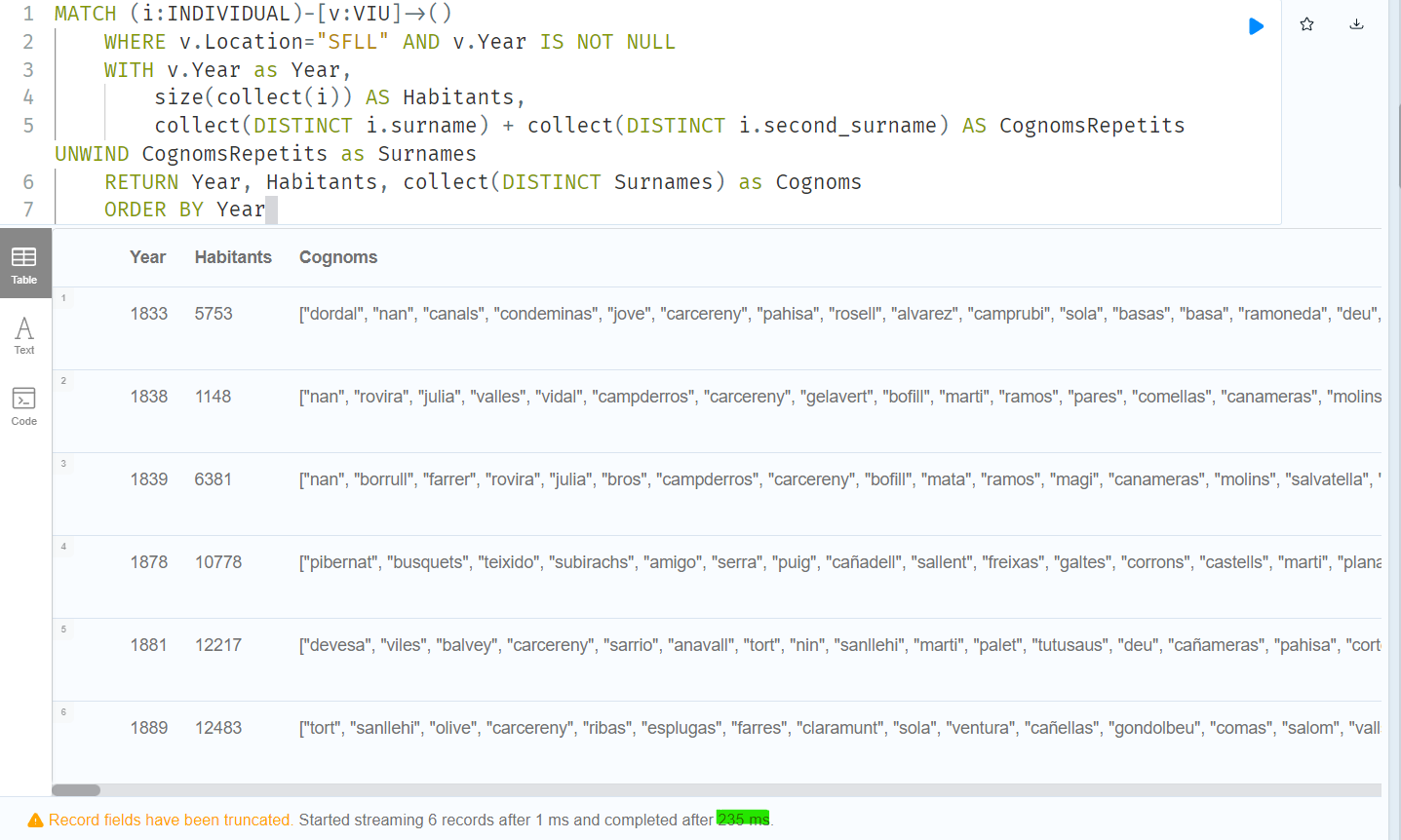


Figura 16



Figura 17

Si executem la consulta B) sense índex, observem com el temps d’execució és de 50ms. En canvi, si ho fem amb l’índex IndividuNomCognoms, aquest decreix als 28ms. Per tant, obtenim una acceleració del . Finalment, concloem que l’índex fa que la consulta s’executi aproximadament 2 vegades més ràpid.

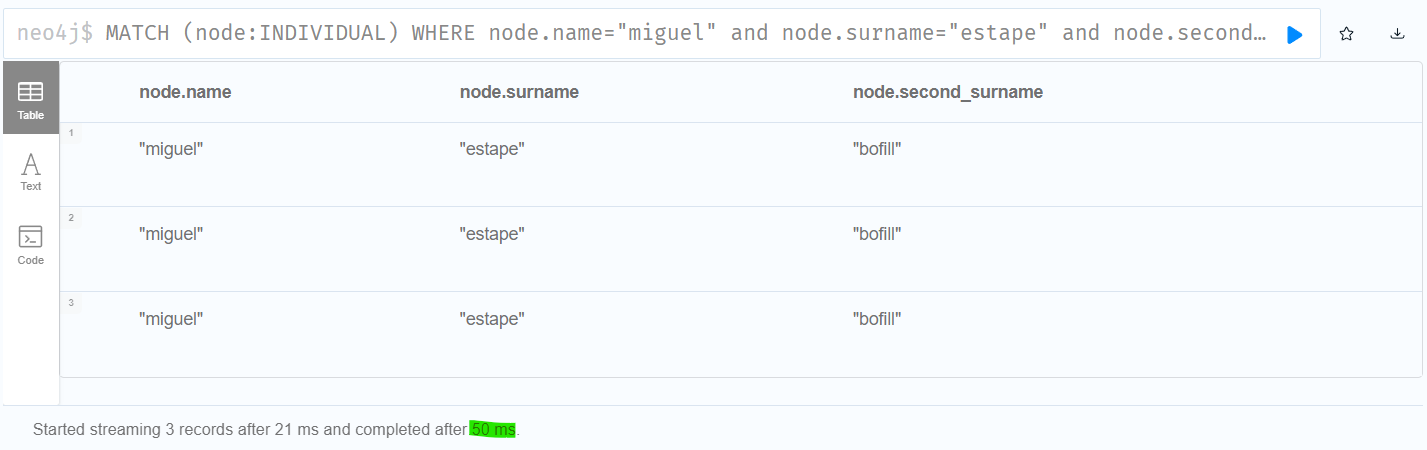


Figura 18

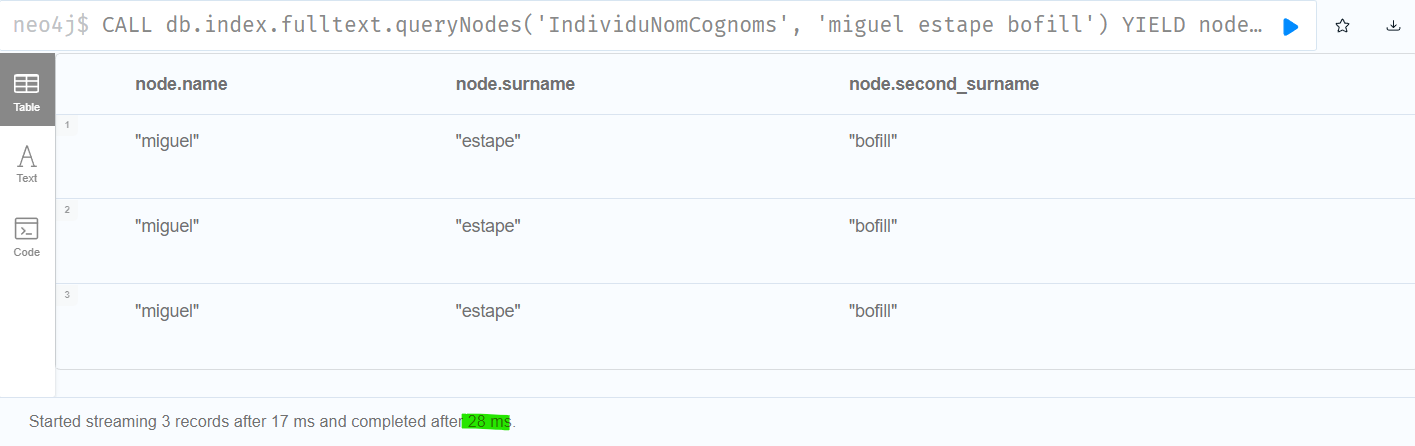


Figura 19

Si executem la consulta C) sense índex, observem com el temps d’execució és de 182ms. En canvi, si ho fem amb l’índex RelacioFamilia, aquest decreix als 21ms. Per tant, obtenim una acceleració del . Finalment, concloem que l’índex fa que la consulta s’executi 8 vegades més ràpid.

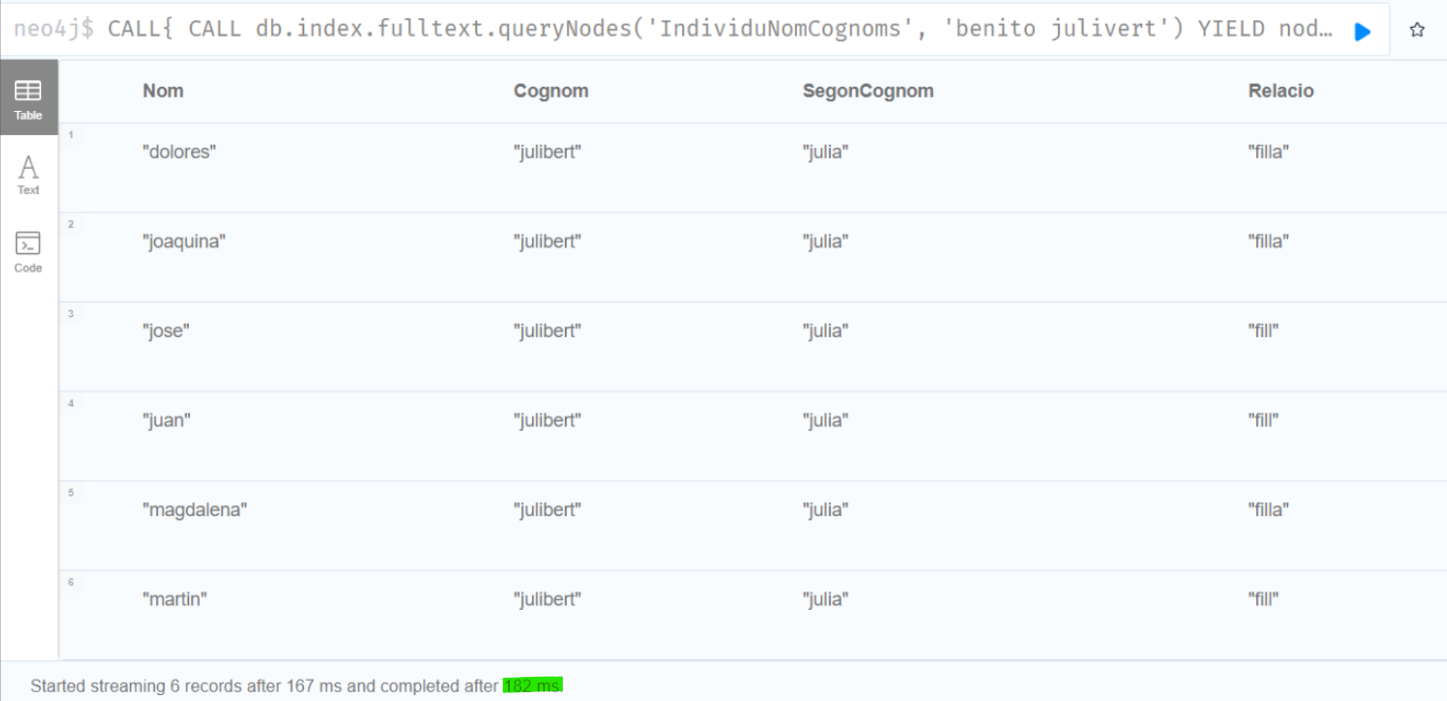


Figura 20

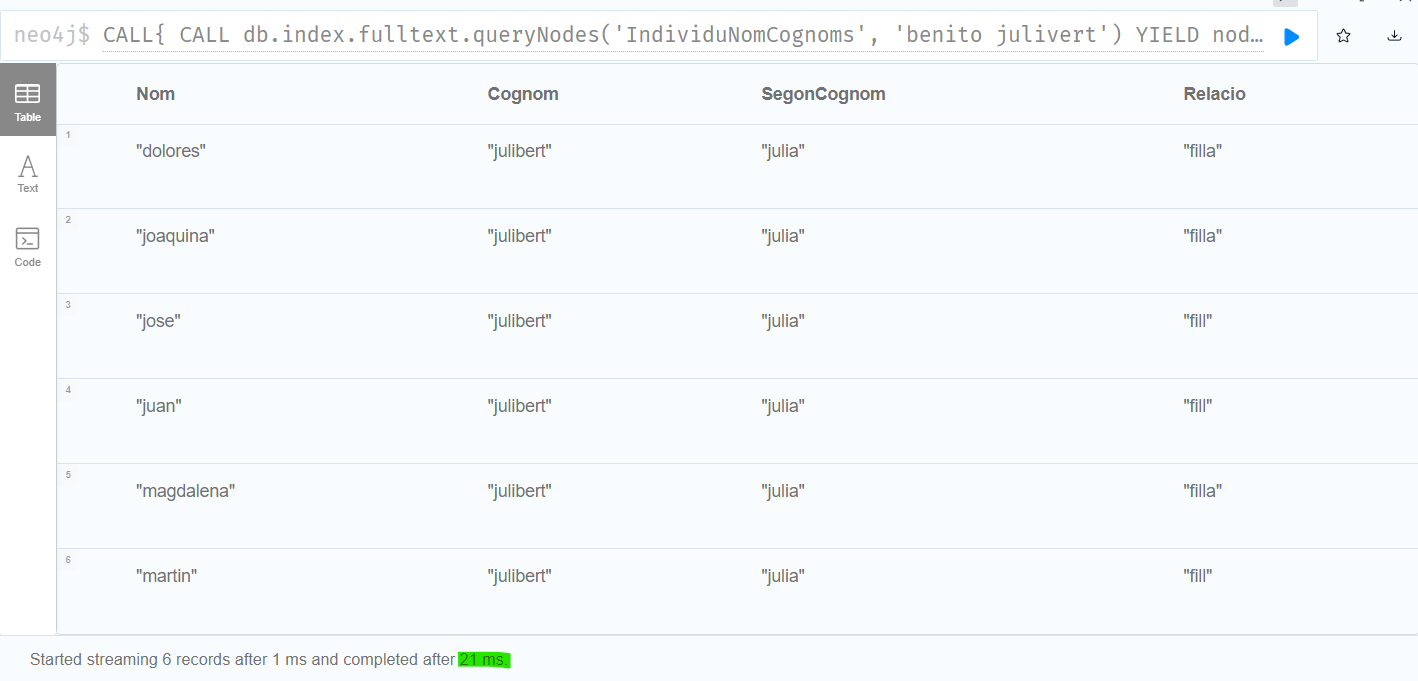


Figura 21

En la següent taula es mostren els speedup obtinguts en l’execució de les consultes amb índex i sense.

Tot i que veiem un notable speedup amb els tres índexs, el major speedup l’ha aconseguit l’índex RelacioFamilia. Això és pel fet que aquest ha estat creat a les arestes FAMILIA, més concretament a la propietat Relacio\_Harmonitzada, el que ens permet accedir molt més ràpidament a aquesta propietat a l’hora de fer la consulta. A més, la consulta C) en la que s’utilitza aquest índex s’està fent una cerca centrada en la propietat RelacioHarmonitzada. Per tant, haver creat aquest índex fa que sigui 8 cops més ràpida.

En canvi, AnyPadroIndex és utilitzat a la consulta A) i ha aconseguit un menor speedup que els altres dos índexs, ja que aquesta consulta no se centra en l’any del padró, sinó que a més ha de contar habitants i els seus cognoms. Per tant, tot i que aquest índex presenta un speedup, no és tan notable com els altres.

Amb l’índex IndividuNomCognoms passa el mateix que AnyPadroIndex. Tot i que aquest presenta un speed-up major que AnyPadroIndex, és molt menor que RelacioFamilia. Un altre cop, l’ús d’aquest índex no és tan notable en la consulta executada.

|  | **AnyPadroIndex** | **IndividuNomCognoms** | **RelacioFamilia** |
| --- | --- | --- | --- |
| **Speed-up** | 1.25x | 1.79x | 8.66x |

Taula 1

## 

## TASCA 3

Analitzeu 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 components en funció de la seva mida. Feu servir el mode stream. Un cop calculades les components connexes (nodes individu, habitatge i relació VIU), feu dues consultes per explorar les dades. Per exemple (podeu fer-ne d’altres):*

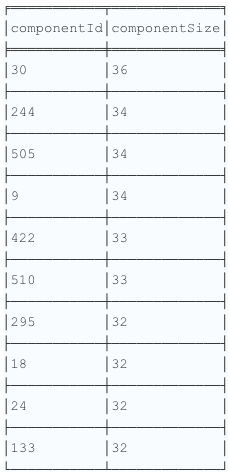
| **CALL gds.graph.project(**  **'grafWCC',**  **['INDIVIDUAL', 'HABITATGES'],**  **{VIU: { type: 'VIU', orientation: 'UNDIRECTED'}})** |
| --- |

Figura 22

Amb aquesta simple comanda agafem tots els nodes individual i habitatges, i agafem les arestes de tipus VIU, i el més important, no-dirigides (en aquest cas obviem les arestes SAME\_AS i FAMILIA, ja que al ser cícliques no ens aporten res al nostre estudi de WCC). Això ho fem per a aconseguir el graf amb arestes unidireccionals, pel fet que si volem calcular les *weakly connected components,* perquè no trobarem cap aresta VIU que apunti d’habitatges a individual, fent que calcular els strongly connected components sigui una cosa sense lògica.

Un cop fet això, apliquem el stream per a calcular les components connexes:

| **CALL gds.wcc.stream('grafWCC')**  **YIELD nodeId, componentId**  **WITH componentId, count(nodeId) AS componentSize**  **RETURN componentId, componentSize**  **ORDER BY componentSize DESC**  **LIMIT 10;** |
| --- |

Figura 23

Un cop cridem al stream de la WCC (weakly connected components), se’ns retornen tant el nodeId, el qual ens dona l’identificador del node, com també un componentId, el qual ens retorna l’identificador de quina component forma part. Per exemple, un node amb *{nodeId: 4, componentId: 1}*, ens estaria indicant que el node 4 forma part de la component connexa 1.

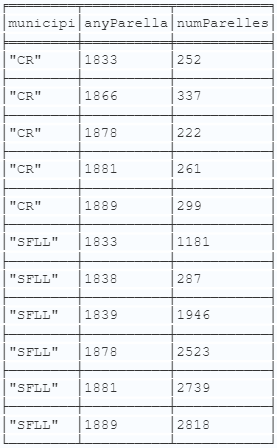
Un cop treta aquesta informació, podem fer un WITH per a comptar els nodeId que hi ha dins de cada componentId, per així poder fer la següent query:

**• Mostra, en forma de taula, les 10 components connexes més grans (ids i mida).**

Figura 24

Si retornem tant el componentId, com el recompte de nodes que hi ha en cada component, i ho ordenem en ordre ascendent limitant-ho només als 10 més grans, obtenim la següent taula (figura 26).

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



| **MATCH (i:INDIVIDUAL)-[r:VIU]->(h:HABITATGES)**  **RETURN h.Municipi as municipi, r.Year as anyParella, count(\*) as numParelles**  **ORDER BY municipi, anyParella ASC;** |
| --- |

Figura 25

D’altra banda, tenim aquesta altra query la qual ens fa un match de la relació INDIVIDU i HABITATGES, i retornem tant el nom del municipi, com l’any, com el nombre de parelles de tipus Individu-Habitatge, la qual ens retorna la taula següent:

Per raons estètiques i de llegibilitat, hem ordenat ascendentment per l’any i pel nom de municipi, per a tenir-ho ordenat cronològicament. Observem doncs, un clar creixement dels individus amb habitatge a Sant Feliu de Llobregat (traient l’anomalia observada al 1838), mentre Figura 26

que a Castellví de Rosanes observem una clara estagnació de la població sobre els 300 individus amb habitatge.

***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). Abans de fer aquesta anàlisi:*

* **Determineu els habitatges que són els mateixos al llarg dels anys. Afegiu una aresta amb nom “MATEIX\_HAB“ entre aquests habitatges. Per evitar arestes duplicades feu que l'aresta apunti a l'habitatge amb any de padró més petit.**

| **MATCH (h1:HABITATGES), (h2:HABITATGES)**  **WHERE h1.Id\_Llar = h2.Id\_Llar**  **AND h1.Any\_Padro < h2.Any\_Padro**  **MERGE (h1)-[:MATEIX\_HAB]->(h2)** |
| --- |

Figura 27

Per realitzar la tasca en qüestió fem servir la query anterior, a la qual busquem tots els habitatges amb el mateix Id\_Llar i els unim amb una aresta amb el nom demanat. A més comprovem quin dels habitatges te un Any\_Pdro més petit per tal de que no hi hagi arestes repetides, ja que fem que l’aresta només apunti al que té l’any més petit.

* **Creeu un graf en memòria que inclogui els nodes Individu i Habitatge i les relacions VIU, FAMILIA, MATEIX\_HAB que acabeu de crear.**

| **CALL gds.graph.project(**  **'subgrafo',**  **['INDIVIDUAL', 'HABITATGES'],**  **{**  **VIU: {**  **type: 'VIU',**  **orientation: 'UNDIRECTED'**  **},**  **FAMILIA: {**  **type: 'FAMILIA',**  **orientation: 'UNDIRECTED'**  **},**  **MATEIX\_HAB: {**  **type: 'MATEIX\_HAB',**  **orientation: 'UNDIRECTED'**  **}**  **}**  **)**  **YIELD graphName, nodeCount, relationshipCount** |
| --- |

Figura 28

Per crear el graf fem un call a la funció graph.project, a la que espicifiquem únicament els tipus de nodes i relacions estipulats a l’enunciat, extraient totes les relacions SAME\_AS entre els nodes INDIVIDUAL.

* **Calculeu la similitud entre els nodes del graf que acabeu de crear, escriviu el resultat de nou a la base de dades i interpreteu els resultats obtinguts.**

| **CALL gds.nodeSimilarity.stream('subgrafo')**  **YIELD node1, node2, similarity**  **WITH gds.util.asNode(node1) AS node1, gds.util.asNode(node2) AS node2, similarity**  **WHERE node1:INDIVIDUAL AND node2:INDIVIDUAL**  **MERGE (node1)-[:SIMILAR\_A {similitud: similarity}]->(node2)** |
| --- |

Figura 29

Per últim, calculem la similitud entre els nodes del graf fent servir la crida nodeSimilarity.stream i escrivim els resultats al graf original com una relació entre nodes amb un atribut numèric al qual anomenem similitud.

Una vegada fetes aquests càlculs i modificacions analitzem els resultats. Primer observem quins nodes tenen una similitud igual a 1 (sent aquests teoricament iguals) amb la següent consulta: (S’ha de considerar que les arestes SIMILAR\_A estan repetides per cada parella de nodes, per tant, s’han de dividir els resultats entre dos).

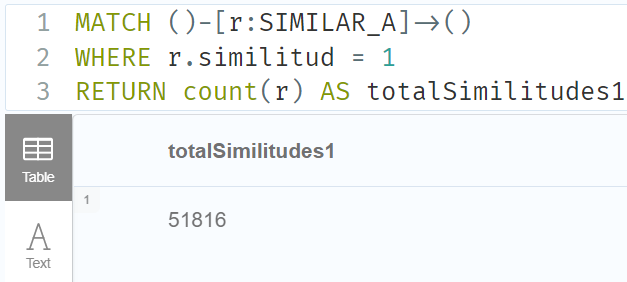


Figura 30

I ara comprovem quins són els nodes relacionats per arestes com SAME\_AS o MATEIX\_HAB, les quals indiquen que els nodes són els mateixos:

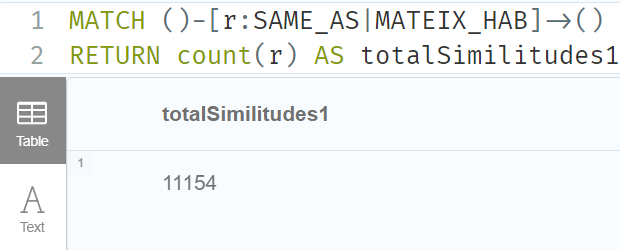


Figura 31

Com poder observar en els resultats de les consultes, la quantitat de nodes teoricament iguals entre ells és més del doble que el nombre de nodes que estan relacionat com iguals entre la base de dades i les transformacions fetes per nosaltres. Això pot ser degut a una falta de integritat de la base de dades proporcionada, la qual no té la suficient informació sobre tots els nodes proporcionats per aquesta. No obstant, si tractem de trobar més nodes iguals amb el paràmetre similitud, augmentant el llindar de tolerància per considerar a dos nodes com “iguals”, no trobarem cap altre relació entre nodes nous fins que no baixem el llindar fins a un 83% de similitud (Consulta i resultats a la següent figura).

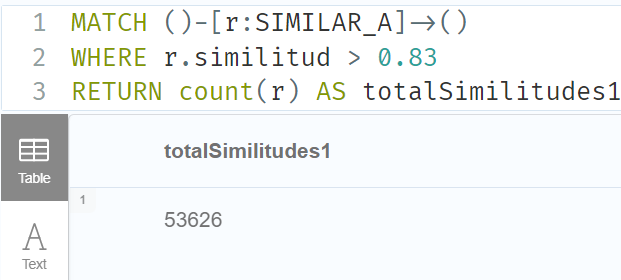


Figura 32

# 

# PART 2

En la part 2 se’ns demanava dissenyar un altre esquema de la base de dades no relacional basada en grafs. Per tal de fer-ho hem hagut d’establir un objectiu per tal de fer el nou disseny. El nostre principal objectiu és millorar l’eficiència per tal de fer unes determinades consultes que s’exposen més endavant.

## TASCA 4

### OBJECTIU I PLANTEJAMENT

En aquesta part del projecte, ens hem proposat redissenyar la base de dades amb l’objectiu de poder veure l’evolució demogràfica de cada poble de manera més eficient. Aquest redisseny implica una nova estructura per organitzar les dades d’acord amb les consultes que es voldran fer, reflectint millor les relacions entre les persones, els seus habitatges i les localitats on viuen. Amb aquestes modificacions, la base de dades serà capaç de proporcionar una comprensió més clara de les dades demogràfiques.

Per tant, un cop decidit l’objectiu vam plantejar una estructura composta per tres tipus de nodes diferents: LOCALITAT, HABITATGE i INDIVIDU.

Els nodes de **LOCALITAT** representaran les diferents àrees geogràfiques, és a dir, els pobles. Aquests nodes tindran com a propietat el Municipi d’on està l’habitatge.

Els nodes **d'HABITATGE** representaran habitatges amb la seva adreça. Aquests tindran com a propietats Id\_Llar (identificador de cada habitatge), Any\_Padro (any de registre de l’habitatge), Carrer (carrer de l’habitatge), Número (número de l’habitatge al carrer) i Municipi (localitat on està l’habitatge).

Finalment, els nodes **d'INDIVIDU** representaran cada persona que residirà en un habitatge determinat dins d'una localitat específica. Aquests tindran com a propietats Id (identificador únic de cada individu), Name (nom de l’individu), Surname i Second\_surname (cognoms de l’individu) i Year (any de naixement de l’individu).

En relació amb les arestes, mantindrem les originals de **FAMILIA i SAME\_AS** amb les seves propietats per conservar la informació relacional que és essencial per identificar les connexions familiars i les equivalències entre nodes. A més a més, afegirem noves arestes per tal de millorar la capacitat de la base de dades per reflectir relacions demogràfiques específiques.

La primera nova aresta que crearem serà **ANY\_VIU**, que connectarà els individus amb els seus corresponents habitatges. Aquesta aresta tindrà com a etiqueta de relació el propi any en el qual va viure el determinat individu a l’habitatge corresponent. Això proporcionarà una dimensió temporal que ens permetrà fer seguiment dels canvis demogràfics al llarg del temps.

La segona nova aresta serà **ANY\_PADRO**, que connectarà cada habitatge amb la seva localitat corresponent. Aquesta aresta ens permetrà situar geogràficament cada habitatge dins de la seva localitat. Per tant, aquesta aresta tindrà com a etiqueta de relació l’any del padró, és a dir, l’any en el qual es va registrar el determinat habitatge amb la corresponent localitat.

Amb aquest redisseny, la base de dades no només millorarà en termes d'organització i estructura, sinó que també ens permetrà dur a terme anàlisis més profundes i acurades de les dades demogràfiques.



Figura 33

Tot i que al redisseny de la base de dades és possible veure redundància de dades, com per exemple l’any padró té la seva justificació.

La redundància de l'any del padró (ANY\_PADRO) en la relació entre els nodes HABITATGE i LOCALITAT i en el node HABITATGE aporta diversos avantatges significatives en la gestió de dades demogràfiques. Primerament, la inclusió d'aquest atribut en la relació entre habitatges i localitats facilita les consultes eficients, ja que permet accedir directament a la informació del padró d'un habitatge sense necessitat de recórrer a altres nodes o relacions. Això millora considerablement la velocitat d'accedir a la informació.

També es pot trobar redundant l’atribut Municipi al node LOCALITAT i HABITATGE. Nosaltres hem volgut crear el node LOCALITAT perquè el nostre objectiu és fer cerques específiques per a cada municipi, per tant, la propietat Municipi del node LOCALITAT ha estat creada per tal de no haver de consultar cada cop el node HABITATGE, el que prediem que reduirà notablement la cerca. No obstant això, si hi ha dues cases amb el mateix Id, Any\_Padro, Carrer i Número a diferents municipis no els podrem diferenciar. És per això que posem el Municipi, per veure que són diferents sense necessitat d'enllaçar l'aresta ANY\_PADRO. D’aquesta manera, podrem fer més eficientment estadístics poblacionals.

A més, tot i que el nostre objectiu no va relacionat amb les relacions entre individus, hem decidit deixar les arestes de FAMILIA i SAME\_AS originals, ja que, per exemple les arestes SAME\_AS són útils per harmonitzar les dades duplicades o coincidents d'individus. En el el nostre cas, la bases de dades conté múltiples entrades per a la mateixa persona, ja sigui a causa d'errors en la introducció de dades, canvis de nom o altres raons. Això millora la integritat i la qualitat de les dades, assegurant que cada individu sigui representat de manera precisa i sense duplicacions. Quant a les arestes FAMILIA, representen les relacions familiars entre individus. Aquesta informació és crucial per comprendre les dinàmiques familiars dins de les localitats i per a altres diverses aplicacions que es podrien implementar dins del nostre nou disseny, tot i que no és el nostre objectiu.

### CREACIÓ I MODIFICACIÓ FITXERS

Una vegada dissenyat el nou esquema, hem plantejat les modificacions a realitzar en la importació de dades:

1. Crear nodes amb label MUNICIPI i <LOCALITAT>.
2. Crear arestes <ANY\_PADRO> i <ANY\_VIU>.

No necessitem canviar la resta d’importacions, ja que els nodes amb label HABITATGES i INDIVIDUAL es mantenen en la BD. D’igual forma que les arestes SAME\_AS i FAMILIA.

Per crear els nodes amb label MUNICIPI i <LOCALITAT>, hem seguit la mateixa metodologia que amb HABITATGES i INDIVIDUAL. És a dir, recollim la informació en un fitxer csv per tenir una visió general de la quantitat de dades que hi ha, i de quin tipus. Per això hem creat un nou arxiu que conté els municipis dels habitatges enregistrats: MUNICIPIS.csv.

Per desenvolupar aquest document, hem utilitzat el llenguatge de programació Python i la llibreria csv. El codi creat (create\_MUNICIPI.py) llegeix el fitxer HABITATGES.csv i s’obté els municipis on són les cases de la BD. Cal destacar que es filtren les dades, és a dir, no es guarda cap localitat amb valor “null”. Tampoc es guarden els municipis més d’una vegada. A continuació, es crea el fitxer MUNICIPIS.csv, indicant la capçalera de l’única columna que contindrà. Seguidament, l’escriptor de la llibreria escriu cada municipi en una fila diferent. Si obrim l’arxiu des d’un Excel, podem observar el contingut de la figura següent.

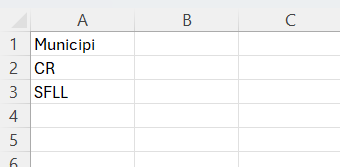


Figura 34

Si observem la taula, només hi ha dos municipis: Sant Feliu de Llobregat i Castellví de Rosanes. Tot i que aquest procés es podria haver fet de forma manual, volem que la nostra implementació sigui escalable i automàtica. És per aquest motiu que hem preferit recolzar-nos en un altre llenguatge de programació. D’altra banda, l’ús de la llibreria csv ha sigut per fer una implementació òptima. En cas de voler afegir més informació, seria tan senzill com afegir una capçalera i les dades corresponents en forma de llista.

Per importar les arestes <ANY\_PADRO> i <ANY\_VIU> no hem creat cap fitxer CSV. Per tal de reduir errors i altres motius, hem decidit crear les arestes directament en el graf. En l’apartat següent explicarem el raonament i procediment.

### IMPORTACIÓ DE DADES

Tal com hem explicat, un dels objectius del nou esquema és millorar l’eficiència de les consultes. Per fer-ho, hem creat nodes i arestes dinàmiques. Un node dinàmic és un node on la seva etiqueta (label) varia en funció de les dades que s’estan processant. Per exemple, els nodes amb l’etiqueta <LOCALITAT> prenen valors com “SFLL” o “CR”. D’igual forma amb les arestes. En el nostre cas, <ANY\_PADRO> i <ANY\_VIU> són arestes dinàmiques. Com en Neo4j els labels no poden començar per valors numèrics, hem afegit el prefix “ANY\_PADRO\_” i “ANY\_VIU\_”, respectivament.

Tot i que Neo4j permet crear nodes i relacions utilitzant Cypher amb paràmetres, les querys tenen algunes restriccions sintàctiques que han de ser respectades. A més, els paràmetres s’han de definir abans d’executar la query. La limitació és que les etiquetes no es poden parametritzar directament en una consulta. El motiu és que Neo4j compila les instruccions prèviament a l’execució. Això és incompatible amb la dinamització de labels perquè aquests canvien durant el temps d’execució i n’hi ha tants que no es poden parametritzar eficientment. Per tant, Neo4j hauria de re-avaluar i re-optimitzar la consulta per cada execució, la qual cosa augmenta significwativament el temps d’execució. Finalment, hem decidit construir les consultes dinàmiques utilitzant el llenguatge de programació Python. Després, establint la connexió amb la base de dades, executem les consultes d’importació i creació d’arestes.

En primer lloc, per crear la BD, importem les dades dels csv originals: HABITATGES.csv, INDIVIDUAL.csv, SAME\_AS.csv i FAMILIA.csv. Això es fa directament des de Neo4j, amb les consultes explicades en la Tasca 1.

En segon lloc, creem els nodes MUNICIPI. Per fer-ho, el fitxer create\_LOCATION.py, obté els labels de <LOCALITAT> llegint les files (de la columna “Municipi”) del document MUNICIPIS.csv. A continuació, estableix la connexió entre Python y la base de dades utilitzant la llibreria neo4j i important les funcions del mòdul GraphDatabase. Després, construeix la consulta Cypher que ens permet crear els nodes amb label MUNICIPI i <LOCALITAT>. Per tant, per cada fila del csv executa la consulta que mostrem a continuació:

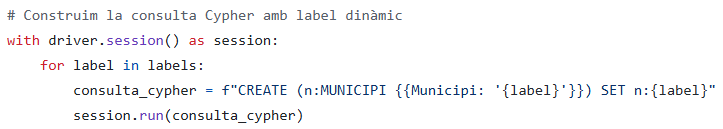


Figura 35

Aquesta consulta crea un node amb l’etiqueta MUNICIPI, aquesta no es modificarà (no es un label dinàmic). A més, s’inicialitza la propietat “Municipi” i s’afegeix una etiqueta amb el valor emmagatzemat en la variable “label” (ara sí és dinàmic). La sintaxis del Cypher és la mateixa que en Neo4j però li afegim f”” al principi per indicar al compilador de Python que ha d’enviar la consulta amb el format indicat. Per acabar, es tanca la connexió.

En tercer lloc, el fitxer create\_ANY\_PADRO.py, obté les dades dels habitatges llegint les files del document HABITATGES.csv. A continuació, estableix la connexió entre Python y la base de dades. Després, construeix la consulta Cypher que ens permet filtrar els nodes MUNICIPI i HABITATGES amb les propietats desitjades. Per cada fila del csv, executa la query que mostrem a continuació:

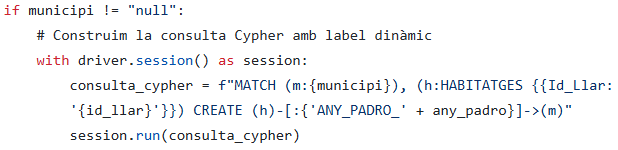


Figura 36

Aquesta consulta crea una aresta amb l’etiqueta <ANY\_PADRO>. Aquesta es crearà si un habitatge pertany a un municipi de la BD. Cal destacar que si les dades del csv contenen el valor “null” no es crearà cap aresta. En altres paraules, filtrem els valors prèviament per garantir la integritat de la BD. Per acabar, es tanca la connexió.

Per últim, el fitxer create\_ANY\_VIU.py, obté les dades dels padrons llegint les files del document VIU.csv. A continuació, estableix la connexió entre Python i Neo4j Després, construeix la consulta Cypher que ens permet filtrar els nodes INDIVIDUAL i HABITATGES amb les propietats desitjades. Per cada fila del csv, executa la consulta que mostrem a continuació:

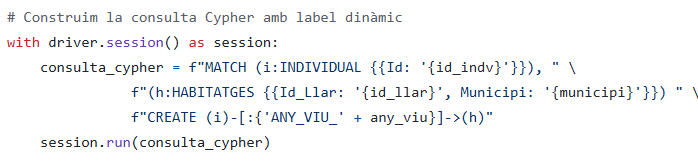


Figura 37

Aquesta consulta crea una aresta amb l’etiqueta <ANY\_VIU>. Aquesta es crearà si una persona ha viscut en un habitatge de la BD. En aquest cas, hem filtrat les dades manualment, és a dir, hem revisat que cap valor del csv fos “null”. Per acabar, es tanca la connexió.

Per tal de reduir la redundància, hem mantingut les CONSTRAINTS creades en la Tasca 1: unique\_id\_person, id\_personExists i id\_llarExists. Tot i que no farem servir les arestes FAMILIA ni SAME\_AS perquè s’allunyen del nostre objectiu, es podria millorar l’accés a les seves propietats fent servir els índexs de la Tasca 1: IndividuNomCognoms i RelacioFamilia. Recordem que aquests proporcionen una acceleració del 1.79x i 8.66x, respectivament.

Un cop importades totes les dades, hi ha 20316 nodes i 80463 arestes. En total, la nostra BD conté 5 node labels, 16 tipus d’arestes i 13 propietats diferents.

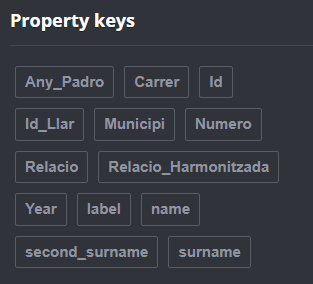
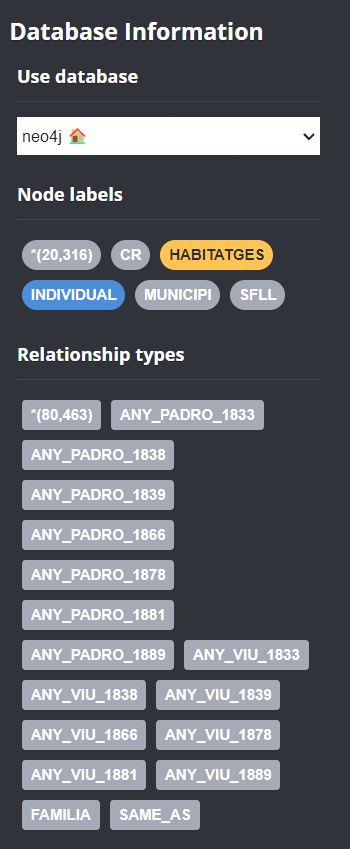
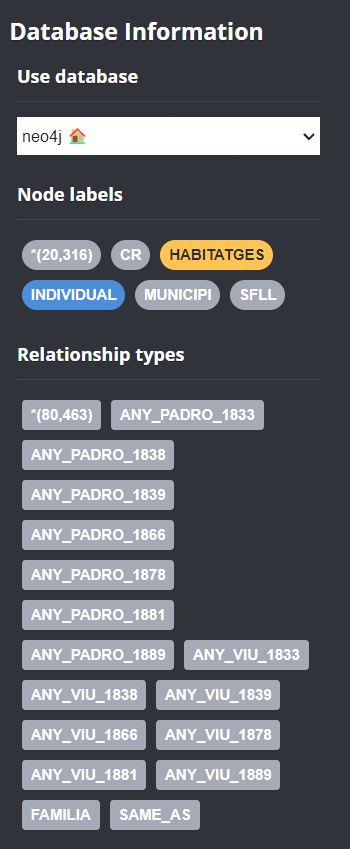


Figura 38, 39 i 40

Per tal de facilitar la importació de dades, hem creat el fitxer instruccio\_execució\_fitxers.csv, ja que l’ordre en que s’introdueixen les dades és important. Si intentem afegir una aresta però els nodes no existeixen, no es crearà l’enllaç correctament. Per tant, en aquest arxiu s’indica quin és l’ordre d’execució que hem de seguir.

### 

### CONSULTES NOU ESQUEMA

Un cop importades les dades i aplicat el nou esquema, podem provar i observar com seran les querys que haviem plantejat per a dur a terme l’estudi demogràfic.

La nostra primera query seguia els següents pragmes:

*Estadístic (per cada any registrat) de l’evolució dels habitatges de cada poble.*

1. *Definir el paràmetre perquè l'usuari introdueixi el poble que vol* ***:param***
2. *Si guardem l'any com una aresta, només hem d'accedir-hi amb type(rel)*

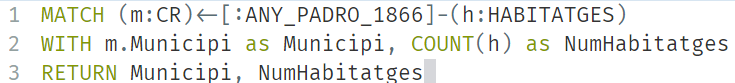
En quant a la segona:

*Estadístic (per cada any registrat) dels cognoms més comuns per cada any.*

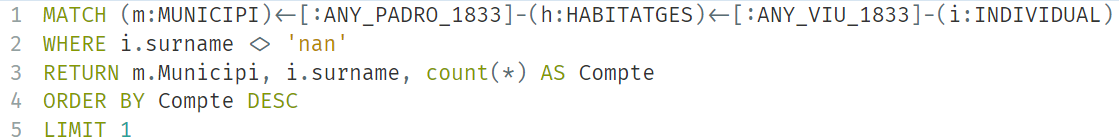
1. *Definir el paràmetre perquè l'usuari introdueixi el poble que vol i l’any* ***:param***
2. *Hi accedim tant amb el type(rel) d’any de padró com el de viu*

Per contratemps amb problemes donats pel funcionament dels plugins *apoc* i *gds*, vam variar les consultes per a fer-les només amb un any, però les comandes estadístiques funcionarien de la mateixa manera donada la escalabilitat de la nostra base de dades.

#### **QUERY 1**



#### **QUERY 2**

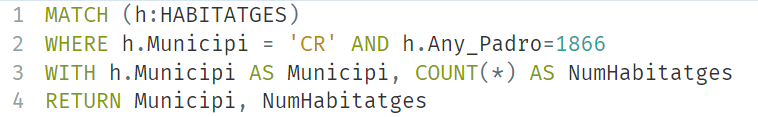


### 

### CONSULTES ANTIC ESQUEMA

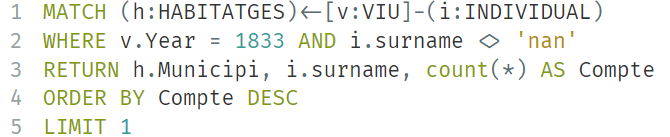
Aquí podem observar com es veurien les nostres querys si ho apliquéssim en la base de dades anterior.

#### **QUERY 1**



En quant a complexitat de la consulta, no observem gaire canvis, tant sols que en comptes de basar-nos en el nostre tipus d’aresta, mirem l’any de padró del propi habitatge.

#### **QUERY 2**



D’altra banda, observem aquí també com en comptes de poder fer servir els nous tipus d’arestes, hem d’accedir a l’any a partir de l’atribut de l’aresta.

### CONSULTES EXTRA

També hem volgut afegir una consulta extra que demostri la escalabilitat que pot arribar a tenir el nostre esquema, tot i que ara només tinguem 2 pobles (donat el prune de NULLs que hem fet). Aquesta consulta es tracta de fer un count del nombre de pobles que hi ha registrats a la nostra base de dades.

En l’esquema **antic** ens quedaria així:

MATCH (h:HABITATGE)

RETURN COUNT(DISTINCT h.Municipi) AS NumMunicipis

Mentrestant, en l’esquema **nou**:

MATCH (m:MUNICIPI)

RETURN COUNT(m) AS NumMunicipis

*Exemple simple però ilustrador d’una avantatge del nostre model.*

1. *Estalviem la necessitat de treballar amb properties i operem amb nodes i labels*
2. *No cal fer l’operació DISTINCT ja que sabem que cada node és únic*

### 

### MILLORA RENDIMENT

Si analitzem els temps d’execució d’ambdues consultes en els dos esquemes (tant l’antic com el nou) en traiem aquests resultats:

|  | **Query 1** | **Query 2** |
| --- | --- | --- |
| **Antic esquema** | ~27.3ms | ~51.4ms |
| **Nou esquema** | ~7.1ms | ~11.7ms |

| **Speed-up** | 3.84x | 4.39x |
| --- | --- | --- |

Aquests valors, obtinguts en mil·lisegons (ms), són fruit de fer la mitjana de 10 execucions en cadascuna, per evitar cap mena d’esbiaixament amb processos externs a la base de dades. Aquestes execucions les hem anat fent alternades, és a dir, primer executàvem la consulta 1, després la 2, i així 10 cops, per evitar cap mena d’esbiaixament donat per la memòria cau del programa.

Com podem observar, la nostra base de dades compleix correctament amb els nostres objectius, esdevinguent més ràpida en totes les comandes que hem provat, per un ampli marge.

En quant a la consulta extra, observem el següent speed-up:

|  | **Query 1** | **Query 2** |
| --- | --- | --- |
| **Temps (ms)** | ~2.2ms | ~17.7ms |

| **Speed-up** | 8,04x |
| --- | --- |

Podem veure doncs, com clarament es demostra que el nostre esquema i la nostra aplicació de la nova base de dades contribueix a una major escalabilitat en quant a estudis demogràfics, ja que observem speed-ups considerables en tots els àmbits.

### 

### AVANTATGES I INCONVENIENTS

La nova base de dades ha estat dissenyada per complir els objectius que es plantejaven des d’un principi, és per això que veiem certs avantatges al nou disseny.

L'estructura és fàcilment escalable, permetent afegir noves localitats, habitatges i individus sense necessitat de redissenyar la base de dades. Si en el futur es necessiten noves relacions (arestes), es poden afegir fàcilment sense afectar l'estructura existent.

També s’ha demostrat que l'estructura permet consultes eficients sobre la població d'una localitat, la composició d'un habitatge o les relacions familiars d'un individu. Aquestes consultes són més ràpides executant-se al nou disseny que a l’anterior. En part gràcies a l’ús de labels en comptes de properties

En relació, l'atribut ANY\_PADRO permet fer seguiment dels canvis al llarg del temps, facilitant l'anàlisi de tendències demogràfiques. Per tant, aquest és un avantatge, ja que el nostre objectiu era fer un estudi demogràfic de cada any.

Com a inconvenients trobem que és important mantenir la base de dades actualitzada amb informació precisa i gestionar les relacions harmonitzades de manera adequada per evitar errors i inconsistències. Per tant, si s’haguessin d’introduir noves dades, tant d’individus com d’habitatges s’hauria d’anar amb cura per tal d’evitar duplicats.

Un altre inconvenient pot ser que amb el creixement de la base de dades, el nombre d'arestes pot augmentar significativament, impactant el rendiment i fent més complicades les operacions de manteniment i consulta. A més, podria ser necessari incrementar els recursos per la nostra base de dades, ja que hauríem d’emmagatzemar molta més informació.