BDNR:
PROJECTE NEO4J

Guió de la resolució dels exercicis del projecte

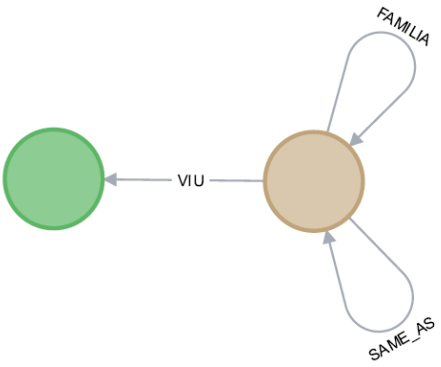
Cristina Huanca (1709896)
Elena Gutiérrez (1703828)
Laia Cámara (1710505)
Laia Alcalde (1642329)

ÍNDEX

Projecte - Part 1.

Suposem que representem la base de dades no relacional basada en grafs de la Figura 3, que conté nodes de tipus habitatges i individus, i tres tipus de relacions:

- **viu**: representen el lloc on viu cada individu.
- **família**: relacions de parentesc entre individus que conviuen al mateix habitatge.
- **same_as**: els nodes que representen el mateix individu al llarg del temps.



Exercici 1.

Importeu les dades en la BD de Neo4j del projecte.
El nostre arxiu script.cypher llueix així:

```
// Creació de restriccions i índexs (Neo4j 5.x+)
CREATE CONSTRAINT municipi_id_unique IF NOT EXISTS
FOR (m:Municipi) REQUIRE m.id IS UNIQUE;

CREATE CONSTRAINT llar_id_unique IF NOT EXISTS
FOR (l:Llar) REQUIRE l.id IS UNIQUE;

CREATE CONSTRAINT individu_id_unique IF NOT EXISTS
FOR (i:Individu) REQUIRE i.id IS UNIQUE;

// Carregar dades de Municipis
LOAD CSV WITH HEADERS FROM 'file:///dades/HABITATGES.csv' AS row
WITH row
WHERE row.Id_Llar <> "null" AND row.Id_Llar <> ""
  AND row.Municipi <> "null" AND row.Municipi <> ""
  AND row.Any_Padro <> "null" AND row.Any_Padro <> ""
  AND row.Numero <> "null" AND row.Numero <> ""
MERGE (m:Municipi {id: row.Municipi})
MERGE (l:Llar {id: toInteger(row.Id_Llar)})
SET l.carrer = row.Carrer,
    l.numero = row.Numero
MERGE (p:Padro {id: row.Municipi + "_" + row.Any_Padro + "_" + row.Id_Llar})
SET p.any = toInteger(row.Any_Padro)
MERGE (p)-[:PERTANY_A]->(m)
MERGE (p)-[:CONTE]->(l);


// Carregar dades de Llars
LOAD CSV WITH HEADERS FROM 'file:///FAMILIA.csv' AS row
WITH row
WHERE row.IdLlar IS NOT NULL
MERGE (l:Llar {id: toInteger(row.IdLlar)})
```

```
SET l.tipus = row.TipusLlar;
// Carregar dades d'Individus
LOAD CSV WITH HEADERS FROM 'file:///dades/INDIVIDUAL.csv' AS row
WITH row WHERE row.Id IS NOT NULL
MERGE (i:Individu {id: toInteger(row.Id)})
SET i.nom = row.name,
    i.cognom = row.surname,
    i.segonCognom = row.second_surname,
    i.any = toInteger(row.Year);


// Crear relacions "VIU" entre Individus i Llars
LOAD CSV WITH HEADERS FROM 'file:///dades/VIU.csv' AS row
WITH row
WHERE row.IND IS NOT NULL AND row.HOUSE_ID IS NOT NULL
MATCH (i:Individu {id: toInteger(row.IND)})
MATCH (l:Llar {id: toInteger(row.HOUSE_ID)})
MERGE (i)-[:VIU]->(l);

// Crear relacions "SAME_AS" entre Individus
LOAD CSV WITH HEADERS FROM 'file:///dades/SAME_AS.csv' AS row
WITH row
WHERE row.Id_A IS NOT NULL AND row.Id_B IS NOT NULL
MATCH (i1:Individu {id: toInteger(row.Id_A)})
MATCH (i2:Individu {id: toInteger(row.Id_B)})
MERGE (i1)-[:SAME_AS]-(i2);
```

En l'script proporcionat, s'utilitzen els següents constraints i índexs:

```
CREATE CONSTRAINT municipi_id_unique IF NOT EXISTS
FOR (m:Municipi) REQUIRE m.id IS UNIQUE;
```

- Propòsit: Garanteix que cada node de tipus Municipi tingui un identificador únic (id).
- Per què?:
    - Evita duplicats en els nodes Municipi.
    - Permet accedir ràpidament a un municipi concret mitjançant el seu id.
    - Millora el rendiment de les operacions que utilitzen MERGE o cerques basades en id.

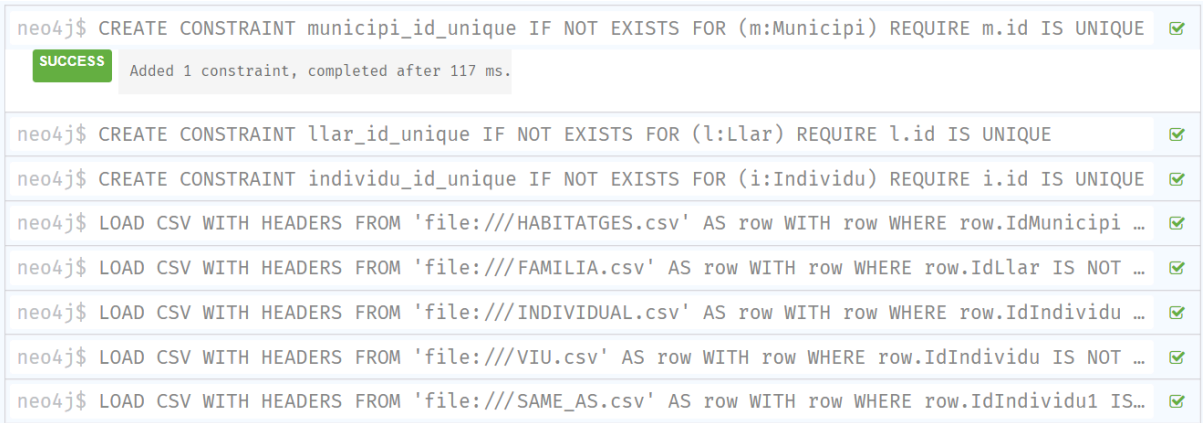
```
CREATE CONSTRAINT llar_id_unique IF NOT EXISTS
FOR (l:Llar) REQUIRE l.id IS UNIQUE;
```

- Propòsit: Garanteix que cada node de tipus Llar tingui un identificador únic (id).
- Per què?:
    - Evita duplicats en els nodes Llar.
    - Facilita la cerca eficient d'una llar específica.
    - Millora el rendiment de les operacions que utilitzen MERGE o cerques basades en id.

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

- Propòsit: Garanteix que cada node de tipus Individu tingui un identificador únic (id).
- Per què?:
    - Evita duplicats en els nodes Individu.
    - Permet accedir ràpidament a un individu concret mitjançant el seu id.
    - Millora el rendiment de les operacions que utilitzen MERGE o cerques basades en id.

Una vegada importades les dades en csv al projecte en Neo4j, carreguem l'script amb èxit:



Exercici 2.

Resoleu les següents consultes Cypher:

a) **Per a cada padró (any) de Castellví de Rosanes, retorna l'any de padró, el número d'habitants, i la llista de cognoms. Elimina duplicats i "nan".**

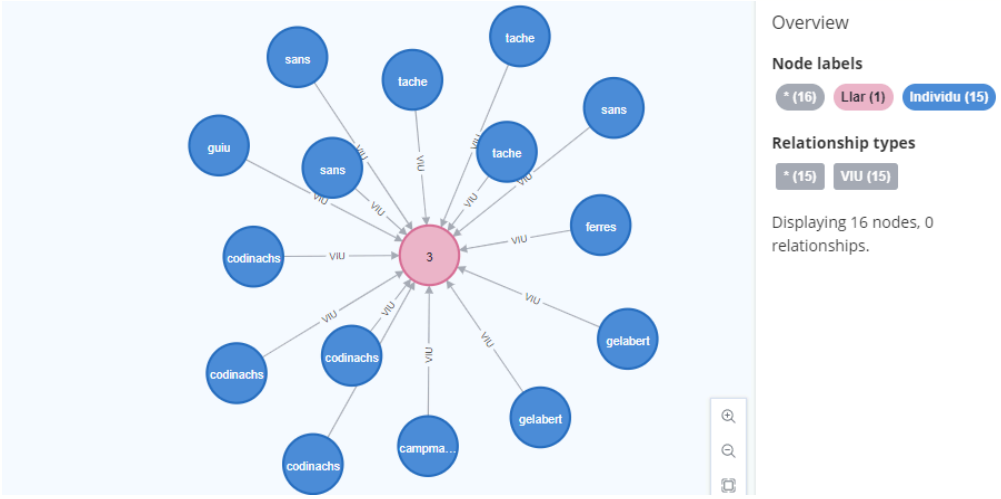
```
MATCH (m:Municipi {id: "CR"})<-[:PERTANY_A]-(p:Padro)-[:CONTE]->(l:Llar)
MATCH (i:Individu)-[:VIU]->(l)
WHERE i.cognom IS NOT NULL AND i.cognom <> "nan"
RETURN p.any AS AnyPadro,
       COUNT(i) AS NumeroHabitants,
       COLLECT(DISTINCT i.cognom) AS LlistaCognoms
ORDER BY AnyPadro;
```

AnyPadro	NumeroHabitants	LlistaCognoms
1866	1182	["galceran", "olle", "galtes", "amigo", "corrons", "tort", "sanllehi", "suñol",

b) **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.**

Coinquilins de "rafel marti" (mode gràfic)

```
MATCH (i:Individu)
WHERE toLower(i.nom) = "rafel" AND toLower(i.cognom) = "marti"
  AND i.any = 1838
WITH i
MATCH (i)-[:VIU]->(l:Llar)<-[:VIU]-(altres:Individu)
WHERE i <> altres
  AND altres.nom IS NOT NULL AND trim(toLower(altres.nom)) <> "nan"
  AND altres.cognom IS NOT NULL AND trim(toLower(altres.cognom)) <> "nan"
   AND altres.segonCognom IS NOT NULL AND trim(toLower(altres.segonCognom))
<> "nan"
RETURN l, collect(altres) AS Coinquilins;
```



Coinquilins de "rafel marti" (mode taula)

```
MATCH (i:Individu)
WHERE toLower(i.nom) = "rafel" AND toLower(i.cognom) = "marti"
  AND i.any = 1838
WITH i
MATCH (i)-[:VIU]->(l:Llar)<-[:VIU]-(altres:Individu)
WHERE i <> altres
  AND altres.nom IS NOT NULL AND trim(toLower(altres.nom)) <> "nan"
  AND altres.cognom IS NOT NULL AND trim(toLower(altres.cognom)) <> "nan"
RETURN DISTINCT
  altres.nom AS Nom,
  altres.cognom AS Cognom,
  altres.segonCognom AS SegonCognom;
```

Nom	Cognom	SegonCognom
"teresa"	"gelabert"	"campmany"
"jose"	"monmany"	"gelabert"
"josefa"	"monmany"	"gelabert"
"pablo"	"calento"	"ferres"
"remigia"	"tache"	"sans"

c) **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 taula: el nom, la llista de cognoms i la llista de segon cognom (elimina duplicats).**

```
MATCH (i:Individu)
WHERE toLower(i.nom) = "miguel"
  AND toLower(i.cognom) = "estape"
  AND toLower(i.segonCognom) = "bofill"
MATCH (i)-[:SAME_AS*]-(altres:Individu)
RETURN DISTINCT
  altres.nom AS Nom,
  COLLECT(DISTINCT altres.cognom) AS LlistaCognoms,
  COLLECT(DISTINCT altres.segonCognom) AS LlistaSegonCognoms;
```

Nom	LlistaCognoms	LlistaSegonCognoms
"miguel"	["estape"]	["bofill", "bufill"]

d) **Mitja de fills a Sant Feliu de 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.**

```
CALL {
        MATCH (m:Municipi {id: "SFLL"})<-[:PERTANY_A]-(p:Padro {any: 1881})-[:CONTE]->(l:Llar)<-[:VIU]-(i:Individu)
  WITH l.id AS id_llar, COUNT(i) AS fillsPerLlar
  RETURN
    SUM(fillsPerLlar) AS TotalFills,
    COUNT(id_llar) AS NombreLlars,
    ROUND(1.0 * SUM(fillsPerLlar) / COUNT(id_llar), 2) AS MitjaFillsPerLlar
}
RETURN *;
```

MitjaFillsPerLlar	NombreLlars	TotalFills
19.44	596	11586

e) **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 (m:Municipi {id: "CR"})<-[:PERTANY_A]-(p:Padro)-[:CONTE]->(l:Llar)
MATCH (i:Individu)-[:VIU]->(l)
WITH l, COUNT(i) AS NumeroFills
WHERE NumeroFills > 3
RETURN
  l.id AS IdLlar,
  l.carrer AS Carrer,
  l.numero AS Numero,
  NumeroFills
ORDER BY NumeroFills DESC
LIMIT 20;
```

IdLlar	Carrer	Numero	NumeroFills
47	"carretera"	"95"	35
18	"carretª"	"13"	33
24	"carretera"	"49"	31
36	"carretera"	"71"	31
51	"carretera"	"992"	28

f) 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 (m:Municipi {id:
"SFLL"})<-[:PERTANY_A]-(p:Padro)-[:CONTE]->(l:Llar)<-[:VIU]-(i:Individu)
  WHERE l.carrer IS NOT NULL AND trim(toLower(l.carrer)) <> "null"
  WITH p.any AS AnyPadro, l.carrer AS Carrer, COUNT(i) AS NumHabitants
  RETURN AnyPadro, Carrer, NumHabitants
}
WITH AnyPadro, Carrer, NumHabitants
ORDER BY AnyPadro, NumHabitants ASC
WITH AnyPadro, collect({carrer: Carrer, habitants: NumHabitants})[0] AS MinCarrer
RETURN
  AnyPadro,
  MinCarrer.carrer AS CarrerAmbMenysHabitants,
  MinCarrer.habitants AS NombreHabitants
ORDER BY AnyPadro ASC;
```

AnyPadro	CarrerAmbMenysHabitants	NombreHabitants
1833	"calle de la carretera"	13
1838	"masover nou"	53
1839	"casas de camp del carrer de dal"	5
1878	"casas de campo del carrer de dal"	12
1881	"falguer"	13

Exercici 3.

Analítica de Grafs: 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)**

Projectem un graf en memòria anomenat viu_graph que conté nodes Individu i Llar, i la relació VIU entre ells. Aquesta relació s'interpreta com no dirigida per permetre identificar components connexes.

```
CALL gds.graph.project(
 'viu_graph',
 ['Individu', 'Llar'],
 {VIU: {
    type: 'VIU',
    orientation: 'UNDIRECTED'
 }});
```

graphName	nodeCount	relationshipCount	projectMillis
"viu_graph"	18351	25730	554

- **Mostra, en forma de taula, les 10 components connexes més grans (ids i mida).** Fem servir l'algorisme Weakly Connected Components (WCC) per identificar quins grups de nodes estan connectats entre si. Retornem les 10 components més grans segons nombre de nodes.

```
CALL gds.wcc.stream('viu_graph')
YIELD nodeId, componentId
RETURN componentId, COUNT(*) AS mida
ORDER BY mida DESC
LIMIT 10;
```

	componentId	mida
1	31	36
2	244	34
3	609	34
4	9	34
5	422	33
6	614	33
7	295	32
8	18	32
9	24	32
10	133	32

- **Quantes components connexes no estan connectades a cap node de tipus 'Habitatge', és a dir, els individus sense casa.**
  Compte quants nodes Individu no tenen cap relació VIU amb un node Llar, és a dir, persones que no estan vinculades a cap domicili censat. Ens ajuda a detectar buits de dades o situacions d'exclusió residencial.

```
MATCH (i:Individu)
WHERE NOT (i)-[:VIU]->(:Llar)
RETURN COUNT(i) AS IndividusSenseLlar;
```

IndividusSenseLlar
4741

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 aquest 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 la aresta apunti al habitatge amb any de padró més petit.**

```
MATCH (l1:Llar)<-[:CONTE]-(p1:Padro),
      (l2:Llar)<-[:CONTE]-(p2:Padro)
WHERE l1.carrer = l2.carrer AND l1.numero = l2.numero
  AND l1 <> l2 AND p1.any < p2.any
MERGE (l1)-[:MATEIX_HAB]->(l2);
```

Creem una nova relació MATEIX_HAB entre habitatges que tenen la mateixa adreça però corresponen a anys de padró diferents. Això permet detectar ocupació d'un mateix domicili al llarg del temps.

```
Created 364 relationships, completed after 2011 ms.
```

- **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(
  'semblanca_graph',
  ['Individu', 'Llar'],
  [
    'VIU',
    'MATEIX_HAB',
    'FAMILIA',
    'SAME_AS'
  ]
);
```

Es crea un graf en memòria amb només els nodes i relacions que ens interessen per a l'anàlisi de semblança.

- **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('semblanca_graph')
YIELD node1, node2, similarity
WHERE similarity > 0.7
WITH gds.util.asNode(node1) AS n1, gds.util.asNode(node2) AS n2, similarity
RETURN n1.nom AS Individu1, n2.nom AS Individu2, similarity
ORDER BY similarity DESC
LIMIT 20;
```

	Individu1	Individu2	similarity
1	*null*	"maria"	1.0
2	*null*	"teresa"	1.0
3	*null*	"eulalia"	1.0
4	*null*	"paula"	1.0

Aquest pas permet guardar les relacions de semblança detectades entre nodes dins del graf semblanca_graph. Per cada parella amb una similitud superior a 0.7, es crea una relació :SEMBLANT_A amb una propietat similitud que en recull el valor. Aquestes relacions es poden fer servir posteriorment per identificar duplicats, explorar agrupacions o fer anàlisis més avançades. El procés retorna quants nodes han estat comparats i quantes relacions han estat escrites. Si el node és una llar, pot aparèixer null a nom, com en el nostre cas.

L'anàlisi de semblança estructural ens proporciona una eina potent per detectar possibles duplicats o perfils similars dins d'un conjunt de dades històriques on la qualitat o consistència de la informació pot variar (ex. noms mal escrits, dates parcials, adreces incompletes).
Les relacions :SEMBLANT_A permeten detectar nodes que tenen un entorn estructural molt similar, com ara individus que viuen a la mateixa llar, comparteixen vincles familiars o repeteixen adreça al llarg dels anys. Aquest tipus de semblança és útil per identificar possibles duplicats o individus que poden ser la mateixa persona en registres diferents. També ajuda a inferir agrupacions familiars o patrons de convivència històrica. L'ús d'aquesta relació és especialment valuós com a pas previ a crear SAME_AS amb fonament estructural.

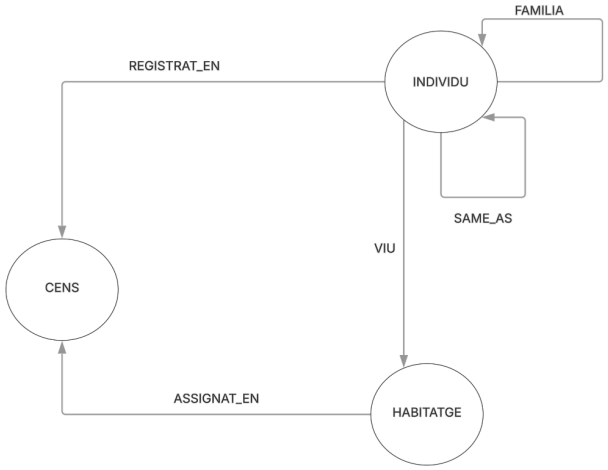
Projecte - Part 2 (optatiu)

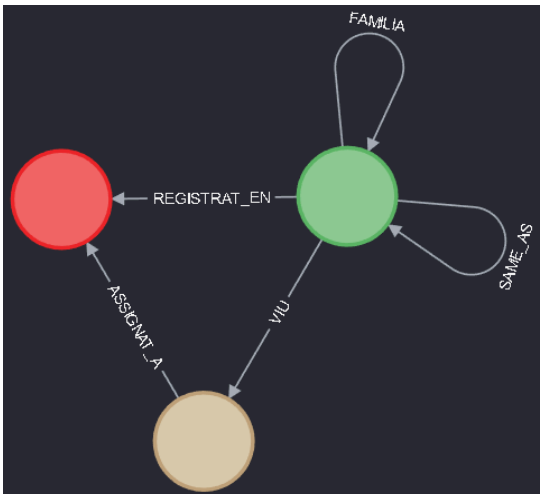
Exercici 4.

Comparativa sobre diferents esquemes de base de dades:
Dissenyeu 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).

```
(:Individu)-[:REGISTRAT_EN]->(:Cens)
(:Habitatge)-[:ASSIGNAT_A]->(:Cens)
(:Individu)-[:VIU]->(:Habitatge)
(:Individu)-[:FAMILIA]->(:Individu)
(:Individu)-[:SAME_AS]->(:Individu)
```





Aquest nou esquema proposat té **3 tipus de nodes:** Individu i Habitatge que era els que ja teníem, i hem afegit el node **Cens,** per representar cada registre de padró. D'aquesta manera, podem agrupar dades sobre periodes de temps i municipis de forma estructurada. Així aconseguim evitar la redundància de dades, millor escalabilitat i eficiència, també aconseguim una major normalització, ja que la informació rellevant es guarda a una entitat pròpia amb relacions específiques.

b) Importeu les dades.

Per importar les dades primer hem de crear els nodes i les seves relacions amb **Cyther**. Destaquem alguns aspectes que fan que la importació sigui més eficient: utilitzem la clàusula **MERGE** per evitar duplicats en elements repetits. El fet que tinguem un esquema més normalitzat, permet aconseguir una millor estructura de dades i importar-les de forma clara evitant columnes redundants o innecessàries.

```
// Garantir que els identificadors són únics
CREATE CONSTRAINT individu_id_unique IF NOT EXISTS
FOR (i:Individu) REQUIRE i.id IS UNIQUE;

CREATE CONSTRAINT habitatge_id_unique IF NOT EXISTS
FOR (h:Habitatge) REQUIRE h.id IS UNIQUE;

CREATE CONSTRAINT cens_unique IF NOT EXISTS
FOR (c:Cens) REQUIRE (c.any, c.municipi) IS NODE KEY;


Crear nodes Cens
LOAD CSV WITH HEADERS FROM 'file:///HABITATGES.csv' AS row
WITH DISTINCT row.any AS any, row.municipi AS municipi
WHERE any IS NOT NULL AND municipi IS NOT NULL
MERGE (:Cens {any: toInteger(row.any), municipi: row.municipi})

Crear nodes Habitatge
LOAD CSV WITH HEADERS FROM 'file:///HABITATGES.csv' AS row
WHERE row.id IS NOT NULL
MERGE (:Habitatge {id: row.id})

Crear nodes Individu
LOAD CSV WITH HEADERS FROM 'file:///INDIVIDUAL.csv' AS row
WHERE row.id IS NOT NULL
MERGE (:Individu {id: row.id, nom: row.nom})

Connectar Habitatge amb Cens
LOAD CSV WITH HEADERS FROM 'file:///HABITATGES.csv' AS row
MATCH (h:Habitatge {id: row.id})
MATCH (c:Cens {any: toInteger(row.any), municipi: row.municipi})
MERGE (h)-[:ASSIGNAT_A]->(c)

Connectar Individu amb Habitatge
LOAD CSV WITH HEADERS FROM 'file:///INDIVIDUAL.csv' AS row
MATCH (i:Individu {id: row.id})
MATCH (h:Habitatge {id: row.id_habitatge})
MERGE (i)-[:VIU]->(h)

Connectar Individu amb Cens
LOAD CSV WITH HEADERS FROM 'file:///INDIVIDUAL.csv' AS row
MATCH (i:Individu {id: row.id})
MATCH (c:Cens {any: toInteger(row.any), municipi: row.municipi})
MERGE (i)-[:REGISTRAT_EN]->(c)
```

El procés es divideix en diverses fases: creació de constraints i índexs per garantir la integritat, inserció dels nodes principals (Individu, Habitatge, Cens), i establiment de les relacions entre ells. A continuació es descriu cada pas acompanyat del codi Cypher utilitzat.

1. Creació de constraints i índexs

Abans d'inserir dades és recomanable definir restriccions d'unicitat per evitar duplicats i millorar el rendiment de les consultes. En el nostre model definim que el node Individu té un identificador únic, igual que el node Habitatge. Per al node Cens, utilitzem una clau composta amb l'any i el municipi.

```
1.Constraints i índexs
CREATE CONSTRAINT individu_id_unique IF NOT EXISTS
FOR (i:Individu) REQUIRE i.id IS UNIQUE;

CREATE CONSTRAINT habitatge_id_unique IF NOT EXISTS
FOR (h:Habitatge) REQUIRE h.id IS UNIQUE;

CREATE CONSTRAINT cens_unique IF NOT EXISTS
FOR (c:Cens) REQUIRE (c.any, c.municipi) IS NODE KEY;
```

2. Creació de nodes Cens

Els nodes de tipus Cens representen cada padró municipal i es defineixen per la combinació de municipi i any. Els extraïm del fitxer HABITATGES.csv i ens assegurem de no duplicar valors utilitzant WITH DISTINCT.

```
2. Creació de nodes
2.1 Nodes Cens
LOAD CSV WITH HEADERS FROM 'file:///HABITATGES.csv' AS row
WITH DISTINCT row.Any_Padro AS any, row.Municipi AS municipi
WHERE any IS NOT NULL AND municipi IS NOT NULL AND any <> "" AND municipi <> "" AND any
<> "null" AND municipi <> "null"
MERGE (:Cens {any: toInteger(any), municipi: municipi});
```

3. Creació de nodes Habitatge

A partir del mateix arxiu es generen els nodes de tipus Habitatge. Cada habitatge té un identificador únic i atributs com el carrer i el número. Es filtra qualsevol valor nul o buit.

```
2.2 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 row.Id_Llar <> "null"
MERGE (h:Habitatge {id: toInteger(row.Id_Llar)})
SET h.carrer = row.Carrer,
    h.numero = CASE
                WHEN row.Numero IS NULL OR row.Numero = "" OR row.Numero = "null"
                THEN null
                ELSE row.Numero
               END;
```

4. Creació de nodes Individu

Els nodes Individu es construeixen a partir del fitxer INDIVIDUAL.csv. Cada persona es defineix pel seu identificador, nom i cognoms, i l'any del padró. També es filtren casos amb identificadors nuls.

```
2.3 Nodes Individu
LOAD CSV WITH HEADERS FROM 'file:///INDIVIDUAL.csv' AS row
WITH row
WHERE row.Id IS NOT NULL AND row.Id <> "" AND row.Id <> "null"
MERGE (i:Individu {id: toInteger(row.Id)})
SET i.nom = row.name,
    i.cognom = row.surname,
    i.segonCognom = row.second_surname,
    i.any = toInteger(row.Year);
```

5. Relació ASSIGNAT_A: Habitatge → Cens

Cada habitatge es vincula amb el cens en què apareix mitjançant la relació ASSIGNAT_A. Aquesta relació permet entendre a quin padró pertany un habitatge concret.

```
3. Creació de relacions
3.1 (:Habitatge)-[:ASSIGNAT_A]->(:Cens)
LOAD CSV WITH HEADERS FROM 'file:///HABITATGES.csv' AS row
WITH row
WHERE row.Id_Llar IS NOT NULL AND row.Id_Llar <> "" AND row.Id_Llar <> "null"
MATCH (h:Habitatge {id: toInteger(row.Id_Llar)})
MATCH (c:Cens {any: toInteger(row.Any_Padro), municipi: row.Municipi})
MERGE (h)-[:ASSIGNAT_A]->(c);
```

6. Relació VIU: Individu → Habitatge

La relació VIU indica on residia cada persona segons el cens. Aquesta informació es treu del fitxer VIU.csv, que conté l'identificador de la persona i de l'habitatge.

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

7. Relació REGISTRAT_EN: Individu → Cens

Amb aquesta relació es modela el fet que una persona està registrada en un cens concret. S'utilitzen els camps any i municipi del fitxer VIU.csv.

```
3.3 (:Individu)-[:REGISTRAT_EN]->(:Cens)
LOAD CSV WITH HEADERS FROM 'file:///VIU.csv' AS row
WITH row
```

```
WHERE row.IND IS NOT NULL AND row.IND <> "null" AND row.Year IS NOT NULL AND
row.Location IS NOT NULL
MATCH (i:Individu {id: toInteger(row.IND)})
MATCH (c:Cens {any: toInteger(row.Year), municipi: row.Location})
MERGE (i)-[:REGISTRAT_EN]->(c);
```

8. Relació FAMILIA: Individu → Individu

Les relacions de parentiu es modelen amb FAMILIA. Quan el tipus de relació no és nul, s'afegeixen propietats; en cas contrari, es crea la relació sense atributs. Es fa servir un doble bloc amb FOREACH per evitar errors amb valors null.

```
3.4 (:Individu)-[:FAMILIA]->(:Individu)
LOAD CSV WITH HEADERS FROM 'file:///FAMILIA.csv' AS row
WITH row
WHERE row.ID_1 IS NOT NULL AND row.ID_1 <> "null"
  AND row.ID_2 IS NOT NULL AND row.ID_2 <> "null"
MATCH (i1:Individu {id: toInteger(row.ID_1)})
MATCH (i2:Individu {id: toInteger(row.ID_2)})
WITH i1, i2, row,
     row.Relacio AS rel,
     row.Relacio_Harmonitzada AS rel_h
FOREACH (_ IN CASE WHEN rel IS NOT NULL AND rel <> "null" AND rel_h IS NOT NULL AND
rel_h <> "null" THEN [1] ELSE [] END |
  MERGE (i1)-[r:FAMILIA]->(i2)
  SET r.relacio = rel, r.relacio_harmonitzada = rel_h
)
FOREACH (_ IN CASE WHEN rel IS NULL OR rel = "null" OR rel_h IS NULL OR rel_h = "null"
THEN [1] ELSE [] END |
  MERGE (i1)-[:FAMILIA]->(i2)
);
```

9. Relació SAME_AS: equivalència entre individus

Aquesta relació es fa servir per identificar casos on dues persones tenen diferents identificadors però es considera que són la mateixa. És una relació simètrica, sense propietats.

```
3.5 (: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_B IS NOT NULL AND row.Id_A <> "null" AND row.Id_B
<> "null"
MATCH (i1:Individu {id: toInteger(row.Id_A)})
MATCH (i2:Individu {id: toInteger(row.Id_B)})
MERGE (i1)-[:SAME_AS]-(i2);
```

c) Feu 2 consultes Cypher (no cal que siguin consultes de l'exercici 2) emprant el nou esquema proposat. Escriviu el codi i el resultat obtingut. Tenint en compte que treballarem amb l'esquema que inclou els nodes:

- (:Individu)
- (:Habitatge)
- (:Cens)

I les relacions:

- (i:Individu)-[:REGISTRAT_EN]->(c:Cens)
- (h:Habitatge)-[:ASSIGNAT_A]->(c:Cens)
- (i:Individu)-[:VIU]->(h:Habitatge)

Les consultes faran ús d'aquestes relacions, sense dependre de cap propietat addicional.

Consulta 1 — Totes les persones registrades al cens de l'any 1866 i municipi "CR"

Aquesta consulta recupera els individus connectats al node Cens amb relació REGISTRAT_EN, filtrant per any i municipi.

```
MATCH (i:Individu)-[:REGISTRAT_EN]->(c:Cens)
WHERE c.any = 1866 AND c.municipi = 'CR'
RETURN i.nom AS Nom, i.id AS ID, c.any AS Any, c.municipi AS Municipi
LIMIT 10;
```

Això en retorna:

	Nom	ID	Any	Municipi
1	"rosa"	18660002	1866	"CR"
2	"antonio"	18660003	1866	"CR"
3	"rosa"	18660004	1866	"CR"
4	"elisa"	18660005	1866	"CR"
5	"manuel"	18660006	1866	"CR"
6	"miguel"	18660007	1866	"CR"
7	"emilia"	18660008	1866	"CR"
8	"isidro"	18660009	1866	"CR"
9	"paula"	18660010	1866	"CR"
10	"jaime"	18660011	1866	"CR"

Consulta 2 — Totes les persones que viuen en habitatges assignats al cens 1881 de “SFLL”

Aquesta consulta segueix la cadena relacional:
(:Individu)-[:VIU]->(:Habitatge)-[:ASSIGNAT_A]->(:Cens)

Filtra pel node Cens corresponent a l’any 1881 i municipi “SFLL”.

```
MATCH (i:Individu)-[:VIU]->(h:Habitatge)-[:ASSIGNAT_A]->(c:Cens)
WHERE c.any = 1881 AND c.municipi = 'SFLL'
RETURN i.nom AS Nom, i.id AS ID, h.id AS HabitatgeID, c.any AS Any,
c.municipi AS Municipi
LIMIT 10;
```

Això en retorna:

	Nom	ID	HabitatgeID	Any	Municipi
1	"rosa"	18660002	1	1881	"SFLL"
2	"antonio"	18660003	1	1881	"SFLL"
3	"rosa"	18660004	1	1881	"SFLL"
4	"elisa"	18660005	1	1881	"SFLL"
5	"manuel"	18660006	1	1881	"SFLL"
6	"miguel"	18660007	1	1881	"SFLL"
7	"emilia"	18660008	1	1881	"SFLL"
8	"jose"	18660001	1	1881	"SFLL"
9	"martin"	18780001	1	1881	"SFLL"
10	"manuela"	18780002	1	1881	"SFLL"

Aquestes consultes reflecteixen perfectament l'estructura del model basat en nodes relacionals. La informació del cens es centralitza en un node independent, fet que evita redundàncies i permet consultes semànticament riques.

d) Feu aquestes mateixes 2 consultes, però ara segons l'esquema de la Figura 3.

Ara replicarem les mateixes consultes utilitzant l'esquema original, que no inclou el node Cens. Per tant:

- No podem fer servir REGISTRAT_EN ni ASSIGNAT_A.
- La informació de municipi i any només es pot inferir a través dels identificadors o està incrustada de manera no explícita (suposada o inexistent).
- Les consultes seran forçosament menys precises i més atributives.

Consulta 1 — Totes les persones registrades al cens de l'any 1866 i municipi "CR"

Com que el node Individu sí que té l'any (gràcies a la columna Year del CSV), però no el municipi, només podríem filtrar per any, però cerquem el node Municipi amb id "CR", trobem els padrons (Padro) d'aquest municipi amb any 1866, d'aquests padrons, obtenim les llars associades i finalment, les persones que viuen en aquestes llars

```
MATCH (i:Individu)-[:VIU_EN]->(h:Habitatge)
WHERE h.any = 1866 AND h.municipi = "CR"
RETURN i.nom AS Nom, i.id AS ID, h.any AS Any, h.municipi AS Municipi
LIMIT 10;
```

Consulta 2 — Totes les persones que viuen en habitatges assignats al cens 1881 de "SFLL"

En el model base, només podem travessar la relació VIU. Com que el node Habitatge no té propietats any ni municipi (i no tenim node Cens), només podem recuperar els individus que viuen en algun habitatge. Així doncs, cerquem el node Municipi amb id = "SFLL", i cerquem el node Padro de l'any 1881 connectat a aquest municipi, trobem les llars associades a aquest padró i finalment, obtenim les persones que viuen en aquestes llars

```
MATCH (i:Individu)-[:VIU_EN]->(h:Habitatge)
WHERE h.any = 1881 AND h.municipi = "SFLL"
RETURN i.nom AS Nom, i.id AS ID, h.any AS Any, h.municipi AS Municipi
LIMIT 10;
```

Inconvenients:

- Si diversos habitatges comparteixen el mateix any i municipi, s'està duplicant la informació en cada node Habitatge.
- Si hi ha un error de municipi en un habitatge, es perd consistència.
- Més difícil d'escalar o de fer consultes per cens en general (per exemple, "quants censos hi ha per municipi?").

Conclusió comparativa:

El nou model permet consultar per censos específics (any + municipi) gràcies a la centralització d'aquesta informació en el node Cens.

El model original, en canvi, no disposa d'aquesta estructura relacional, i les consultes es basen només en atributs individuals, amb menys capacitat de filtratge.

Per tant, el model nou és molt més potent per fer consultes analítiques, mentre que el model base és més simple però limitat en escenaris reals.

e) Raoneu els avantatges i inconvenients de l'esquema proposat en comparació al model proposat a la Figura 3.

Un **avantatge** de l'esquema que hem proposat en comparació al model original és que en incorporar cens com a entitat independent aconseguim la **normalització de dades**, d'aquesta forma aconseguim organitzar de forma adequada la informació per evitar redundàncies, assegurant consistència i integritat de les dades. Encara que això té com a inconvenient que les consultes perden la seva simplicitat, però en utilitzar *MATCH* amb relacions i nodes simples fa que sigui **més eficients**.

En conclusió, els canvis realitzats es podrien considerar tots com a avantatges, ja que millora l'escalabilitat i eficiència. Tot i això, aquests canvis poden requerir passos addicionals a l'hora d'importar les dades per normalitzar i connectar les entitats.

```
Esquema de la Figura 3 (original)

Nodes:
:Individu
:Habitatge

Relacions:
(:Individu)-[:VIU]->(:Habitatge)
(:Individu)-[:FAMILIA]->(:Individu) (autorealació)
(:Individu)-[:SAME_AS]->(:Individu) (autorealació)


Esquema Proposat

Nodes:
:Individu
:Habitatge
:Cens (afegits)

Relacions:
(:Individu)-[:REGISTRAT_EN]->(:Cens)
(:Habitatge)-[:ASSIGNAT_A]->(:Cens)
(:Individu)-[:VIU]->(:Habitatge)
(:Individu)-[:FAMILIA]->(:Individu)
(:Individu)-[:SAME_AS]->(:Individu))
```

Treball en equip

Ens hem dividit el treball en les 4 persones que som, al principi com no enteníem molt bé si s'havia de fer commits, vam anar posant cadascuna la seva part del codi en un document compartit, llavors quan ho vam entendre la Cristina Huanca es va dedicar a pujar tots els commits.

Cristina Huanca: Investigació ex. 4 + Commits de Github i gestió de codi dels primers dos exercicis
Laia Alcalde: Investigació ex. 4 + Començar presentació i document (justificar codis) i justificar exercici optatiu (amb Laia Cámara)
Elena Gutiérrez: Investigació ex. 4 + Exercici tres i ajudar a Cristina amb primers dos exercicis.
Laia Cámara: Investigació ex. 4 + Polir presentació i document, commit document (justificar codis) i justificar exercici optatiu (amb Laia Alcalde)

Adreça al repositori de GitHub:
https://github.com/criisshg/Neo4j_1709896