Rapport

Partie 1 : Données

1) Origine des données :

La collection de données que nous utilisons pour notre projet a été générée par le site Food.com. Elle se compose essentiellement des fichiers ***RAW_interactions.csv*** et ***RAW_recipes.csv*** et est en téléchargement libre sur le site Kaggle.com après création d'un compte. Pour notre projet nous l'avons également rendu disponible sans condition de téléchargement, via le lien suivant : https://drive.google.com/file/d/1MH4_9OQfAekBI8lmVfyg4e_ypqXA9YC2/view?usp=drive_link

2) Contexte du jeu de données :

Notre jeu de données est une collection de plus de 180000 recettes de cuisine et de plus de 700000 revues sur ces recettes. C'est 18 ans d'interactions utilisateur et de téléchargements de recette sur le site Food.com (anciennement GeniusKitchen).

3) Prétraitement des données :

Afin de rendre notre jeu de données pertinent et optimisé pour notre projet, nous allons effectuer un prétraitement avec le fichier ***script.py*** ramassé avec ce rapport. Les étapes de ce prétraitement sont :

- Charger des fichiers de la collection

```python
import pandas as pd

# Chemins des fichiers d'entrée
recettes_file = "RAW_recipes.csv"
interactions_file = "RAW_interactions.csv"

# Chargement des fichiers
recettes = pd.read_csv(recettes_file)
interactions = pd.read_csv(interactions_file)

```

- Filtrer les recettes : on ne gardera que les recettes qui ont plus de dix revus

```python
# Définir les seuils pour filtrage
min_interactions_recette = 10  # Garder les recettes avec au moins 10 interactions
top_percentage_users = 0.1   # Garder les interactions des 10% des utilisateurs les plus actifs

# Filtrer les recettes populaires
recette_counts = interactions['recipe_id'].value_counts()

popular_recettes = recette_counts[recette_counts >= min_interactions_recette].index
filtered_recettes = recettes[recettes['id'].isin(popular_recettes)]

```

- Filtrer les revus : on ne gardera que les revus des utilisateurs les plus actifs

```python
# Filtrer les utilisateurs actifs
user_counts = interactions['user_id'].value_counts()
top_users = user_counts.nlargest(int(len(user_counts) * top_percentage_users)).index
filtered_interactions = interactions[
    (interactions['recipe_id'].isin(popular_recettes)) &
    (interactions['user_id'].isin(top_users))
]

```

- Enlever les colonnes qui ne sont pas utiles à notre analyse :

```python
# Supprimer la colonne 'description' après le filtrage des recettes
if 'description' in filtered_recettes.columns:
    filtered_recettes = filtered_recettes.drop(columns=['description'])

# Supprimer la colonne 'steps' après le filtrage des recettes
if 'steps' in filtered_recettes.columns:
    filtered_recettes = filtered_recettes.drop(columns=['steps'])
```

- Enfin nous sauvegardons les données traitées dans les différents fichiers ***filtered_recettes.csv*** et ***filtered_interactions.csv***

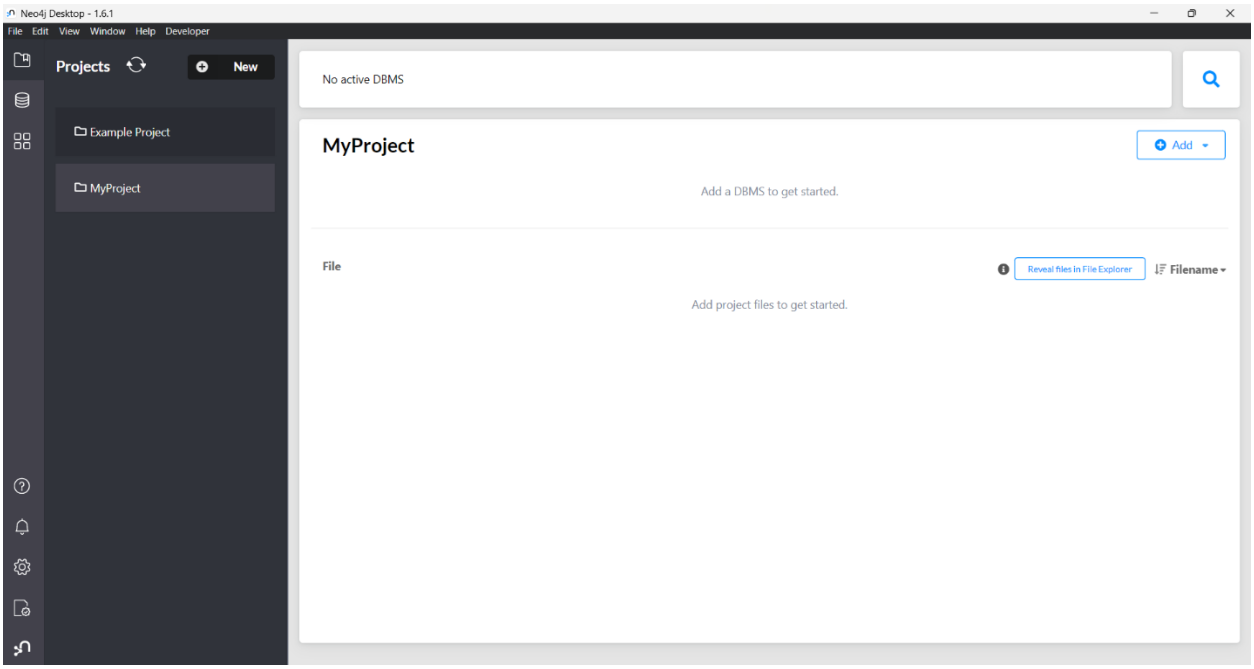
```
46  # Sauvegarder les fichiers réduits
47  filtered_recettes.to_csv("filtered_recettes.csv", index=False)
48  filtered_interactions.to_csv("filtered_interactions.csv", index=False)
49
50  print("Fichiers réduits sauvegardés.")
51
```

Partie 2 : Chargement dans Neo4j

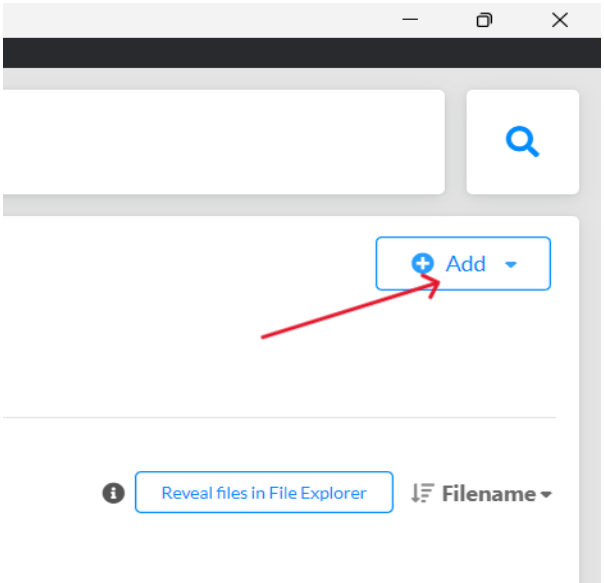
1) Les données que nous chargeons dans Neo4j sont ceux des fichiers ***filtered_recettes.csv*** et ***filtered_interactions.csv.***

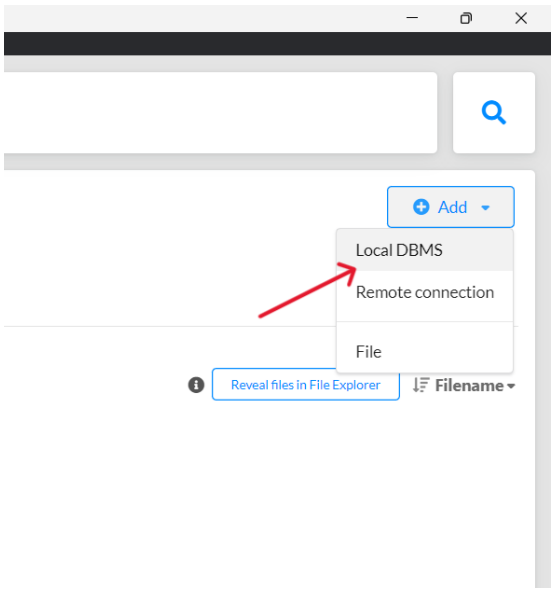
2) Lors du chargement des données :
    - Nous allons transformer les données recettes en nœud Recette
    - L'attribut tag en nœud Tag qui sera lié à Recette avec une relation HAS_TAG.
    - L'attribut ingrédient sera aussi un nœud et sera relié à Recette avec la relation USES.
    - Les interactions vont devenir des nœuds User reliés à recette par la relation RATED.
    - Nous créerons aussi le nœud CategorieRecette qui sera liée à Recette par la relation BELONGS_TO. Les catégories de recette sont : « **Entrée** », « **Plat principal** », « **Dessert** » et « **Autre** ». Pour pouvoir catégoriser les recettes, nous allons nous baser sur les tags les plus pertinents, ces tags ayant une importance plus grande pour nous dans notre recommandation *('appetizers', 'main-dish', 'desserts',…)*

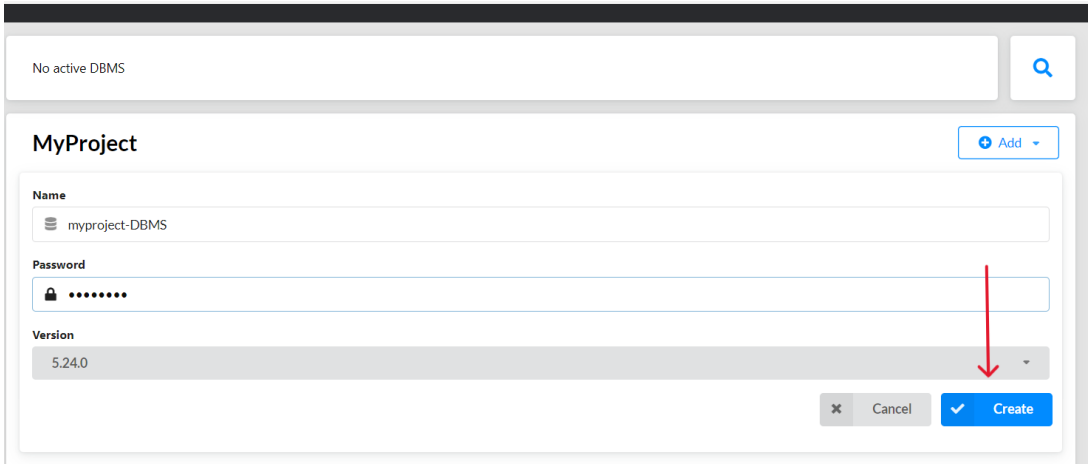
3) Pour charger nos données, nous avons les étapes suivantes :
    - **Etape 1** : Nous allons ouvrir Neo4j Desktop et sélectionnons le projet cible sinon nous en créons un nouveau. Notre projet s'appelle dans ce cas « **MyProject** »

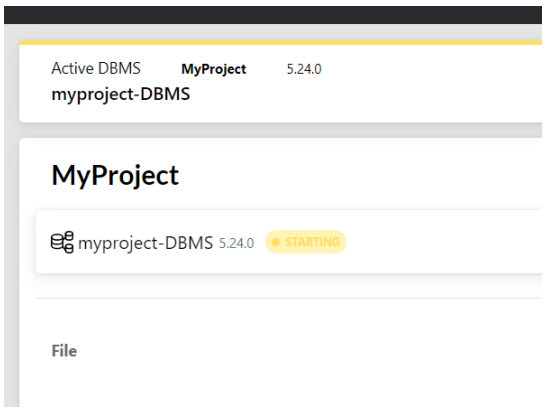


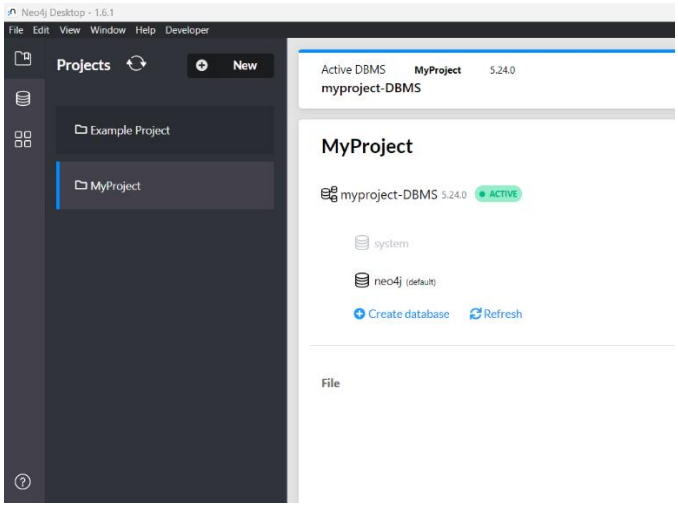
- **Etape 2** : Nous créons un nouveau DBSM (Database Management System) « **myproject-DBSM »** qui va nous aider dans la gestion de la base de données graphe et nous la démarrons.




Add
Local DBMS
Remote connection
File
Reveal files in File Explorer
Filename

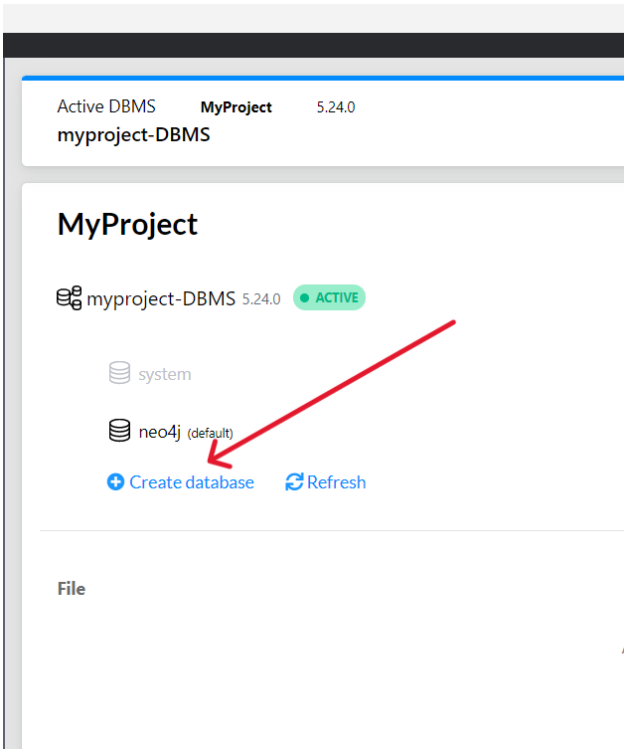

No active DBMS
MyProject
Add
Name
myproject-DBMS
Password
Version
5.24.0
Cancel
Create

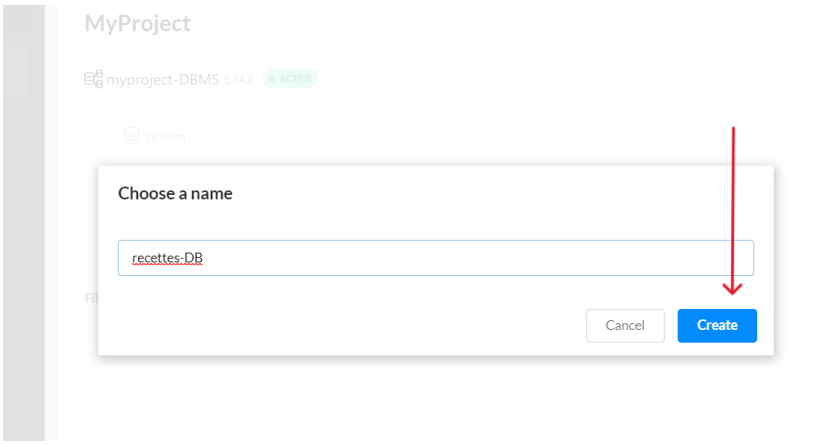

Active DBMS
MyProject
5.24.0
myproject-DBMS
MyProject
myproject-DBMS 5.24.0
STARTING
File

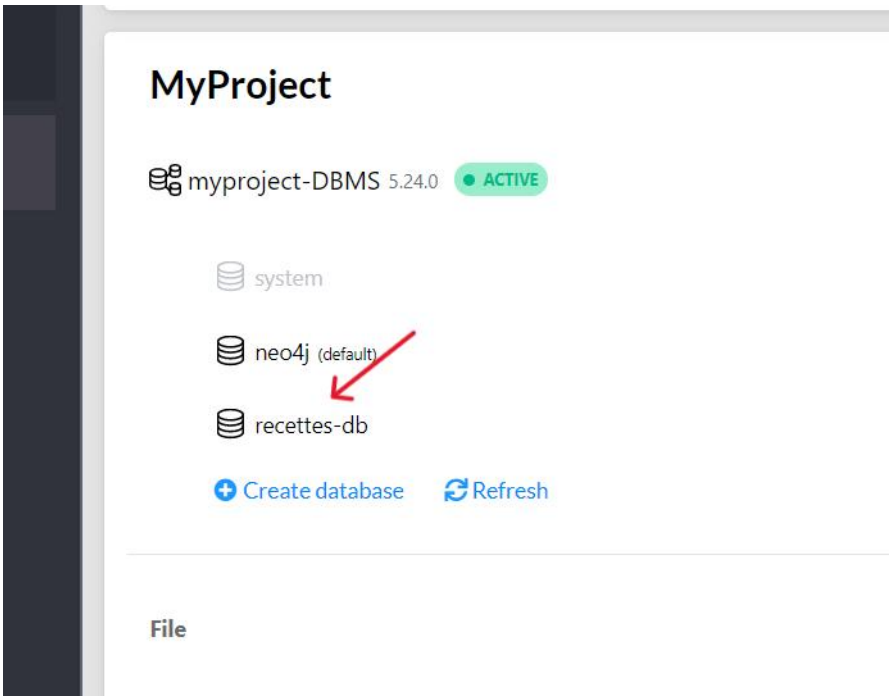


- **<u>Etape 3</u>** : Créer la base de données « **recettes-db** »

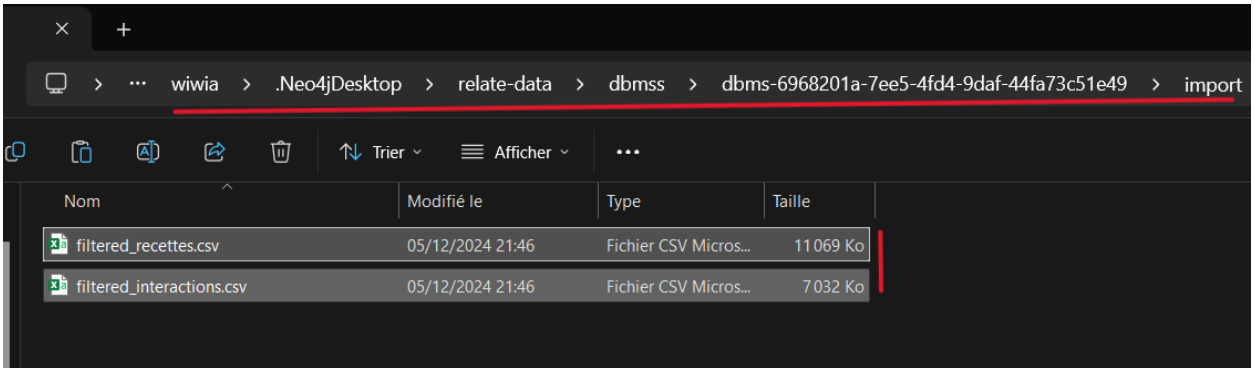
-



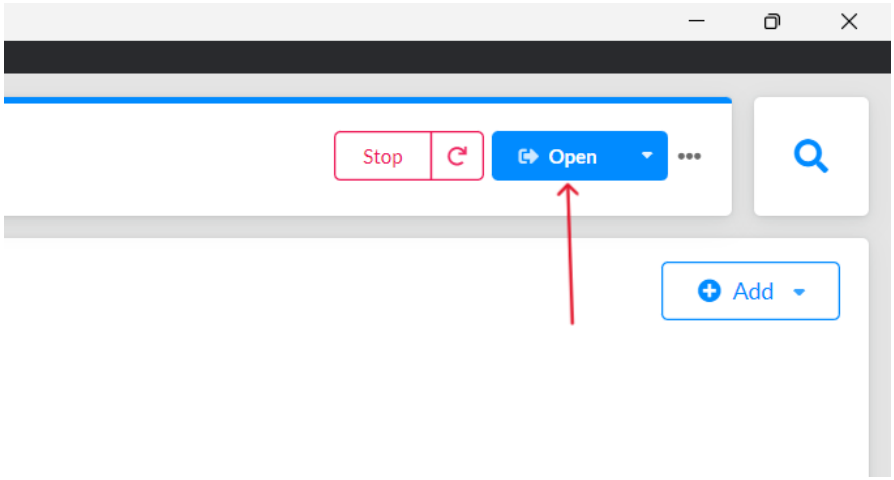


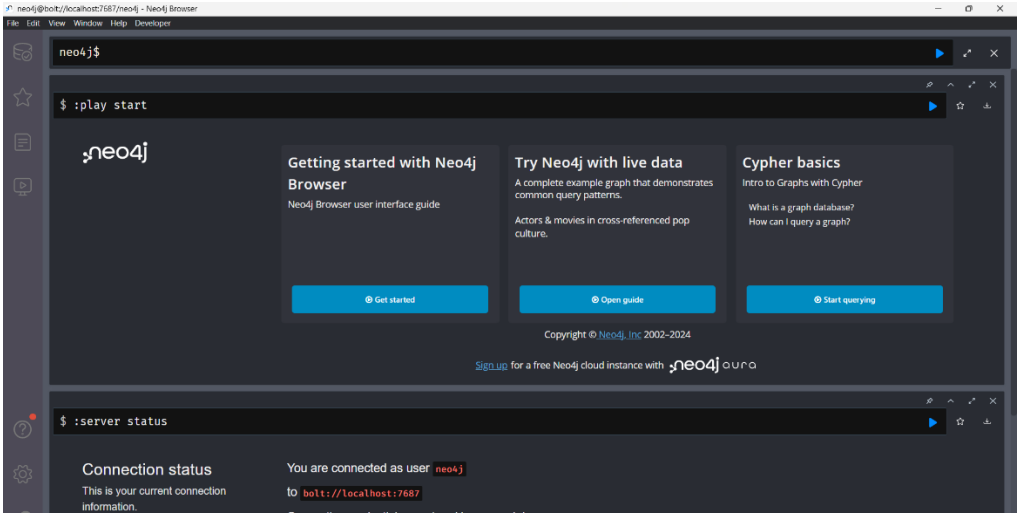


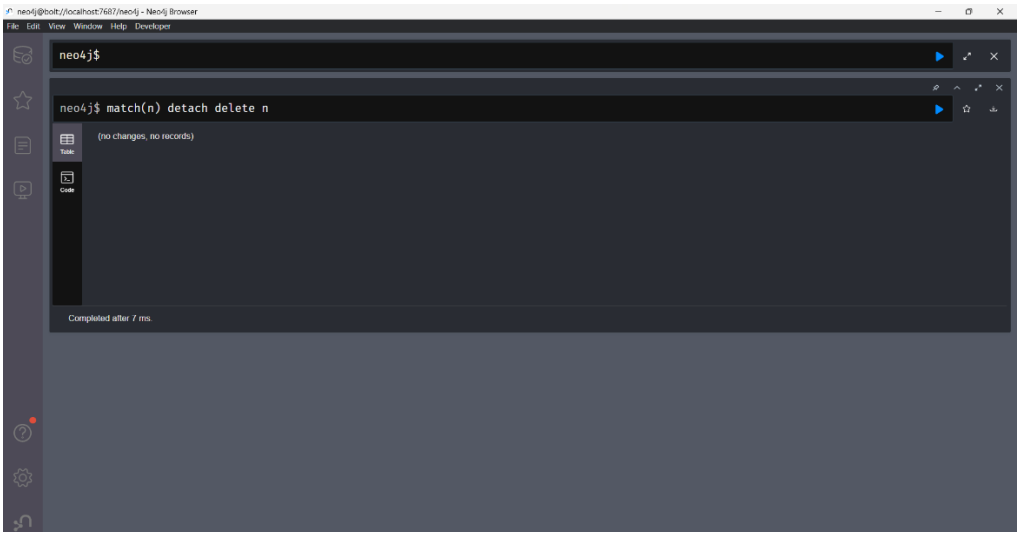
- **<u>Etape 4</u>** : Copier nos fichiers dans le répertoire par défaut de « **myproject-DBMS** » afin de pouvoir les exploiter.



- **<u>Etape 5</u>** : Ouvrir la console Neo4j. Nous pouvons nous assurer que la base de données soit bien vide en entrant la commande `match(n) detach delete n` comme suit





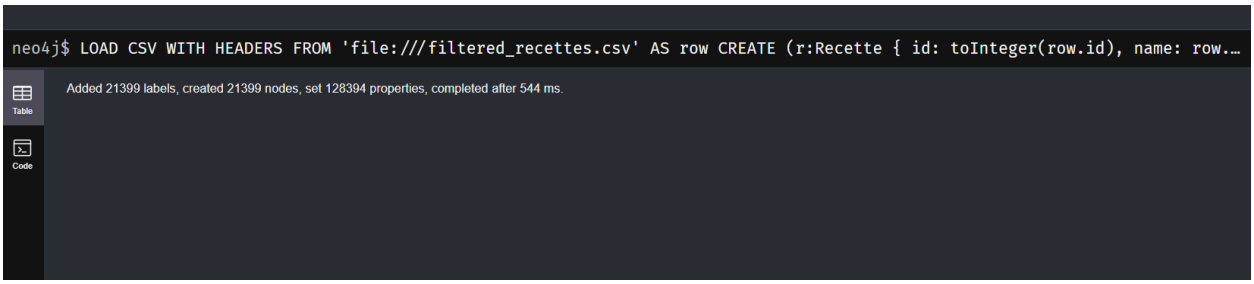


- **Etape 6** : Nous créons les index avant le chargement pour optimiser les requêtes :

```
CREATE INDEX FOR (r:Recette) ON (r.id);
CREATE INDEX FOR (t:Tag) ON (t.name);
CREATE INDEX FOR (i:Ingredient) ON (i.name);
```

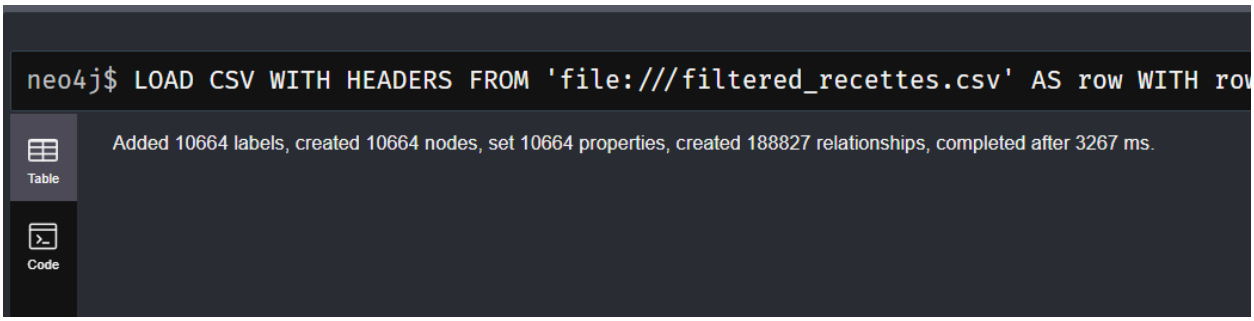
- **Etape 7** : Nous chargeons les recettes avec un modèle Recette Le code de chargement est le suivant :

```
LOAD CSV WITH HEADERS FROM 'file:///filtered_recettes.csv' AS row
CREATE (r:Recette {
            id: toInteger(row.id),
          name: row.name,
          minutes: toInteger(row.minutes),
           contributor_id: toInteger(row.contributor_id),
          submitted: date(row.submitted),
           n_ingredients: toInteger(row.n_ingredients)
    });
```



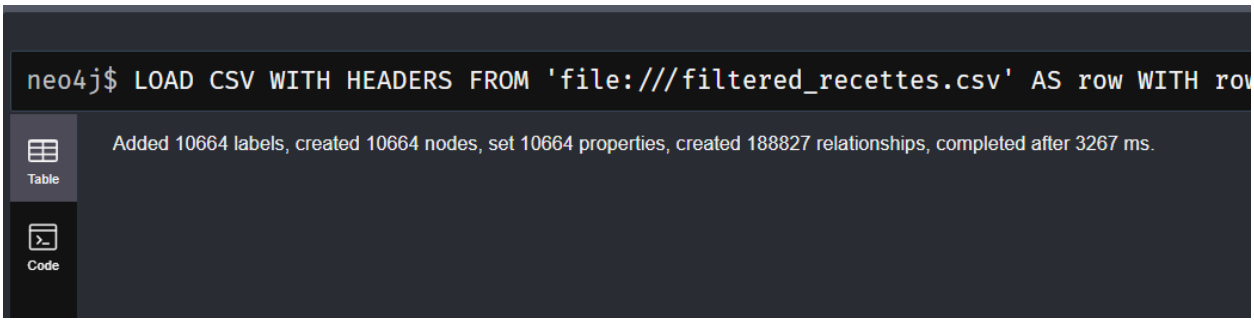
- **<u>Etape 8</u>** : Nous chargeons les tags des recettes et créons la relation HAS_TAG entre recette et tag.
  Code :

```
LOAD CSV WITH HEADERS FROM 'file:///filtered_recettes.csv' AS row
WITH row, split(replace(row.tags, "'", ""), ", ") AS tags
UNWIND tags AS tag
MERGE (t:Tag {name: trim(tag)})
WITH row, t
MATCH (r:Recette {id: toInteger(row.id)})
MERGE (r)-[:HAS_TAG]->(t);
```



- **<u>Etape 9</u>** : Nous chargeons les ingrédients des recettes et créons la relation USES entre recette et ingrédient.
  Code :

```
LOAD CSV WITH HEADERS FROM 'file:///filtered_recettes.csv' AS row
WITH row, split(replace(row.ingredients, "'", ""), ", ") AS ingredients
UNWIND ingredients AS ingredient
MERGE (i:Ingredient {name: trim(ingredient)})
WITH row, i
MATCH (r:Recette {id: toInteger(row.id)})
MERGE (r)-[:USES]->(i);
```



- **<u>Etape 10</u>** : On charge les utilisateurs et leur revus et on crée la relation RATED entre recette et user.
  Code:

```
LOAD CSV WITH HEADERS FROM 'file:///filtered_interactions.csv' AS row
MERGE (u:User {user_id: toInteger(row.user_id)})
    WITH u, row
MATCH (r:Recette {id: toInteger(row.recipe_id)})

MERGE (u)-[rel:RATED {rating: toInteger(row.rating)}]->(r);
```

- **<u>Etape 11</u>** : On crée les catégories et ensuite on les lie aux recettes
  Code

```
  CREATE (c:CategorieRecette {name: 'Entrée', tags: ['appetizers', 'side-
dishes', 'snacks']});
  CREATE (c:CategorieRecette {name: 'Plat principal', tags: ['main-
dish', 'side-dishes', 'meat', 'lunch']});
  CREATE (c:CategorieRecette {name: 'Dessert', tags: ['desserts', 'frozen-
desserts', 'snacks', 'kid-friendly']});
  CREATE (c:CategorieRecette {name: 'Autre', tags: []});


  MATCH (r:Recette)-[:HAS_TAG]->(t:Tag), (c:CategorieRecette {name: 'Entrée'})
  WHERE t.name IN c.tags
  MERGE (r)-[:BELONGS_TO]->(c);
```

```
MATCH (r:Recette)-[:HAS_TAG]->(t:Tag), (c:CategorieRecette {name: 'Dessert'})
WHERE t.name IN c.tags
MERGE (r)-[:BELONGS_TO]->(c);
```

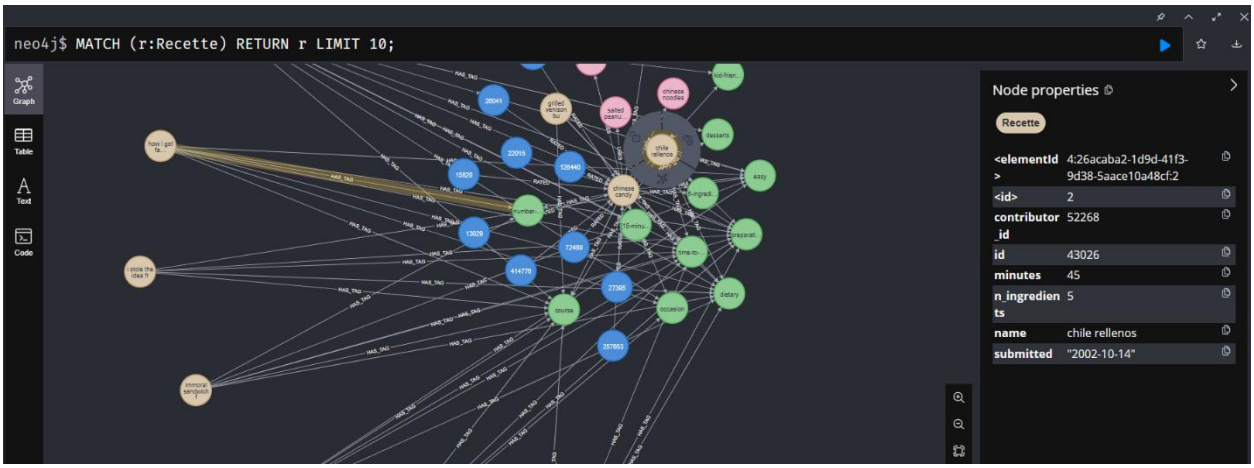
```
MATCH (r:Recette)-[:HAS_TAG]->(t:Tag), (c:CategorieRecette {name: 'Plat
principal'})
WHERE t.name IN c.tags
MERGE (r)-[:BELONGS_TO]->(c);
```

```
MATCH (r:Recette)
WHERE NOT (r)-[:BELONGS_TO]->(:CategorieRecette)
MERGE (r)-[:BELONGS_TO]->(c:CategorieRecette {name: 'Autre'});
```

- **Etape 12** : On vérifie si les données sont bien enregistrées avec les relations.

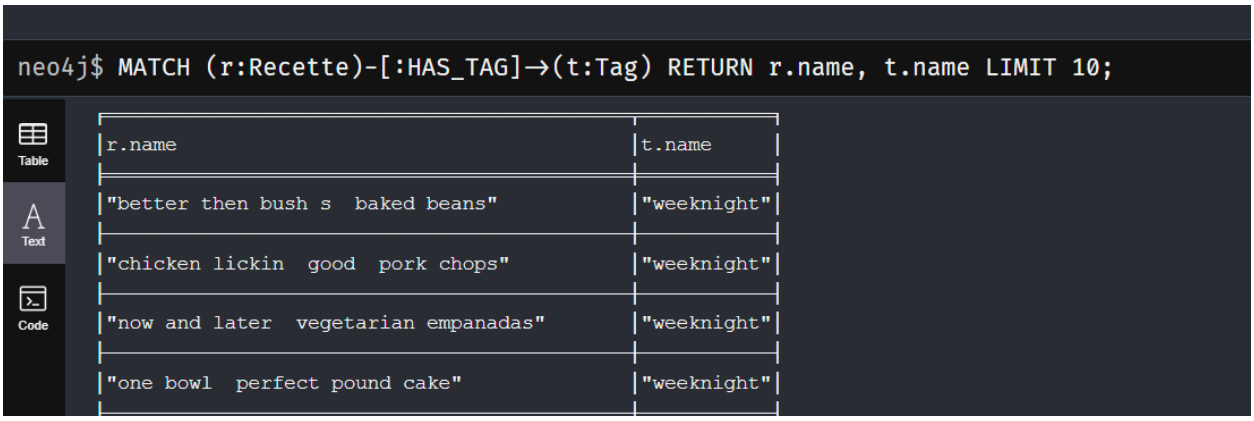
Code pour lister les recettes :

```
MATCH (r:Recette) RETURN r LIMIT 10;
```



Code pour lister les relations avec les tags :

```
MATCH (r:Recette)-[:HAS_TAG]->(t:Tag) RETURN r.name, t.name LIMIT 10;
```



Code pour lister les relations avec les ingrédients

```
MATCH (r:Recette)-[:USES]->(i:Ingredient) RETURN r.name, i.name LIMIT 10;
```

```
neo4j$ MATCH (r:Recette)-[:USES]→(i:Ingredient) RETURN r.name, i.name LIMIT 10;
```

Table | Text | Code

r.name	i.name
"better then bush s baked beans"	"[great northern bean"
"southern great northern beans crock pot version"	"[great northern bean"
"u s senate bean soup"	"[great northern bean"
"better then bush s baked beans"	"chicken bouillon cubes"
"panera bread black bean soup"	"chicken bouillon cubes"
"amy s potato soup crock pot or stove top"	"chicken bouillon cubes"
"awesome chili"	"chicken bouillon cubes"
"bacon and potato chowder"	"chicken bouillon cubes"

Code pour lister les relations avec les users

```
MATCH (u:User)-[rel:RATED]->(r:Recette)
RETURN u.user_id, r.id, rel.rating
LIMIT 10;
```

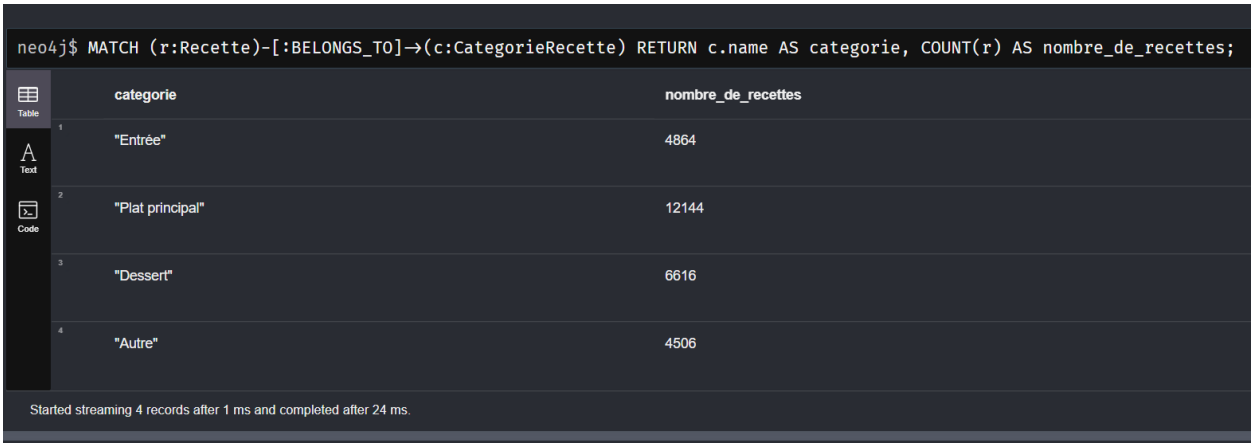
```
neo4j$ MATCH (u:User)-[rel:RATED]→(r:Recette) RETURN u.user_id, r.id, rel.rating LIMIT 10;
```

Table | Text | Code

u.user_id	r.id	rel.rating
118163	67547	5
23479	67547	4
742955	67547	2
28649	63986	4
95743	63986	5
60992	63986	5
371738	63986	5
250238	63986	4

Code pour lister les catégories de recette

```
MATCH (r:Recette)-[:BELONGS_TO]->(c:CategorieRecette)
RETURN c.name AS categorie, COUNT(r) AS nombre_de_recettes ;
```



```
neo4j$ MATCH (r:Recette)-[:BELONGS_TO]→(c:CategorieRecette) RETURN c.name AS categorie, COUNT(r) AS nombre_de_recettes;
```

Table | Text | Code

	categorie	nombre_de_recettes
1	"Entrée"	4864
2	"Plat principal"	12144
3	"Dessert"	6616
4	"Autre"	4506

Started streaming 4 records after 1 ms and completed after 24 ms.

Partie 3 : Recommandations

1) Pour notre projet nous allons faire une recommandation d'un plat à un client dans le cadre d'une application de restauration en ligne par exemple**.**

   Plus concrètement sur la base d'un plat (recette) que le client (user) a déjà commandé plus d'une fois (on supposera qu'il l'apprécie donc), nous allons lui faire des recommandations de plats susceptible de l'intéresser. Cette recommandation reposera sur les catégories, les tags, la pertinence des ingrédients et de la similarité de Pearson avec les autres utilisateurs.

2) La requête de recommandation comprend :

- Calculer de la similarité de Pearson : Nous calculons la similarité de Pearson entre l'utilisateur source (u1), le client et les autres utilisateurs (u2), en utilisant leurs revus.
  Les utilisateurs similaires sont identifiés par la moyenne de leur note et le calcul de la similarité.

```
 MATCH (u1:User {user_id: 56680})-[r:RATED]->(r1:Recette)
 WITH u1, avg(r.rating) AS u1_mean
 MATCH (u1)-[r1_rating:RATED]->(r1:Recette)<-[r2_rating:RATED]-(u2:User)

WITH u1, u1_mean, u2, COLLECT({r1_rating: r1_rating, r2_rating: r2_rating}) AS
ratings WHERE size(ratings) > 10
 MATCH (u2)-[r2:RATED]->(r2_node:Recette)
 WITH u1, u1_mean, u2, avg(r2.rating) AS u2_mean, ratings
 UNWIND ratings AS r

WITH sum( (r.r1_rating.rating - u1_mean) * (r.r2_rating.rating - u2_mean) ) AS
x1,
sqrt( sum( (r.r1_rating.rating - u1_mean)^2) * sum( (r.r2_rating.rating - u2_me
an) ^2)) AS x2,
u1, u2 WHERE x2 <> 0
 WITH u1, u2, x1/x2 AS pearson
```

- Récupérer la catégorie du plat source en utilisant la relation BELONGS_TO entre la Recette et la CategorieRecette ce qui affinera notre recommandation

```
  MATCH (r1:Recette {id: 26835})-[:BELONGS_TO]->(cat:CategorieRecette)
  WITH cat.name AS category, u2
```

- Trouver les k recettes les mieux notées de la même catégorie que la recette source par chaque User similaire. Les recettes sont ensuite triées en fonction de leur note moyenne, et par similitude des tags et des ingrédients avec la recette source.

```
  MATCH (u2)-[r:RATED]->(r2:Recette)-[:BELONGS_TO]-
>(cat:CategorieRecette {name: category})
  WHERE r.rating > 3
  WITH r2, AVG(r.rating) AS average_rating
  MATCH (r2)-[:HAS_TAG]->(tag:Tag)<-[:HAS_TAG]-(r1:Recette {id: 26835})
  WITH r2, average_rating, COUNT(tag) AS common_tags
  MATCH (r2)-[:USES]->(ingredient:Ingredient)<-[:USES]-(r1)

WITH r2, average_rating, common_tags, COUNT(ingredient) AS common_ingredients
  ORDER BY average_rating DESC, common_tags DESC, common_ingredients DESC
  LIMIT 10
```

- Calculer un score personnalisé : le score final des recommandations est un produit de la similarité de Pearson et de la note moyenne de la recette. Cela rajoute de la précision quant aux recettes des utilisateurs qui partagent des préférences proches de celles du client.

```
  WITH r2, average_rating, common_tags, common_ingredients, pearson
RETURN r2.name AS recommended_recipe, average_rating, common_tags, common_ingre
dients,
       pearson * average_rating AS weighted_score
```

3) Requête complete avec recette_id: 26835 et user_id: 56680

```
 MATCH (u1:User {user_id: 56680})-[r:RATED]->(r1:Recette)
 WITH u1, avg(r.rating) AS u1_mean
 MATCH (u1)-[r1_rating:RATED]->(r1:Recette)<-[r2_rating:RATED]-(u2:User)
WITH u1, u1_mean, u2, COLLECT({r1_rating: r1_rating, r2_rating: r2_rating}) AS
ratings WHERE size(ratings) > 10
 MATCH (u2)-[r2:RATED]->(r2_node:Recette)
 WITH u1, u1_mean, u2, avg(r2.rating) AS u2_mean, ratings
 UNWIND ratings AS r
```

```
WITH sum( (r.r1_rating.rating - u1_mean) * (r.r2_rating.rating - u2_mean) ) AS
x1,
sqrt( sum( (r.r1_rating.rating - u1_mean)^2) * sum( (r.r2_rating.rating - u2_me
an) ^2)) AS x2,
u1, u2 WHERE x2 <> 0
 WITH u1, u2, x1/x2 AS pearson


// 2. Trouver la catégorie de la recette source
 MATCH (r1:Recette {id: 26835})-[:BELONGS_TO]->(cat:CategorieRecette)
 WITH cat.name AS category, u2, pearson

// 3. Trouver les 10 meilleures recettes de la même catégorie que la recette so
urce
 MATCH (u2)-[r:RATED]->(r2:Recette)-[:BELONGS_TO]-
>(cat:CategorieRecette {name: category})
 WHERE r.rating > 3
 WITH r2, AVG(r.rating) AS average_rating,pearson
 MATCH (r2)-[:HAS_TAG]->(tag:Tag)<-[:HAS_TAG]-(r1:Recette {id: 26835})
 WITH r2, average_rating, COUNT(tag) AS common_tags,pearson
 MATCH (r2)-[:USES]->(ingredient:Ingredient)<-[:USES]-(r1)
WITH r2, average_rating, common_tags, COUNT(ingredient) AS common_ingredients,p
earson
 ORDER BY average_rating DESC, common_tags DESC, common_ingredients DESC
 LIMIT 10

// 4. Calculer un score personnalisé pour chaque recette recommandée basé sur l
a similarité des utilisateurs
 WITH r2, average_rating, common_tags, common_ingredients, pearson
RETURN r2.name AS recommended_recipe, average_rating, common_tags, common_ingre
dients,
       pearson * average_rating AS weighted_score
 ORDER BY weighted_score DESC
 LIMIT 10;
```

Résultats:

	recommended_recipe	average_rating	common_tags	common_ingredients	weighted_score
2	"butterscotch apple pecan cobbler"	5.0	22	8203	2.703448996979118
3	"the best chocolate chip cookies ever"	5.0	21	20116	2.2919230856165225
4	"mexican tres leche cake"	5.0	22	12593	1.6305792755959392
5	"italian love cake super easy"	5.0	23	13969	-0.030060857480304386
6	"bisquickie cinnamon rolls"	5.0	21	13177	-0.4171864304033489 6
7	"dump cake only 3 ingredients"	5.0	22	605	-0.47694813852512474

ted streaming 10 records after 51 ms and completed after 979 ms.