Projet NoSql (partie II)

-

Rapport de Alizé Baudin

Le 17 mai 2023

III. Redis

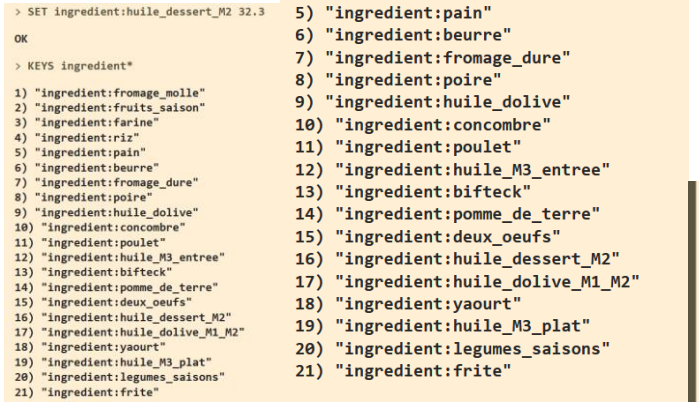
Pour résoudre cette problématique sous Redis, nous utilisons la structure de données clé-valeur de Redis pour stocker les informations nécessaires sur les ingrédients, les plats et les menus, ainsi que leurs empreintes carbones respectives.

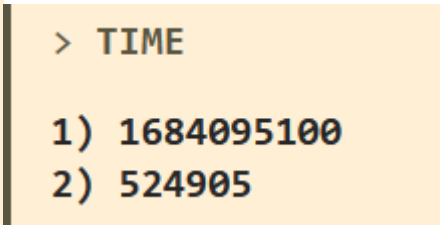
1. Stocker l'empreinte carbone de chaque ingredient :

On utilise ici une clé Redis pour chaque ingrédient, par exemple :

SET ingredient:<nom> <empreinte_carbone>

En particulier pour stocker plusieurs données en même temps au lieu de 'SET' on utilise 'MSET' comme nous le voyons ci-dessous :





```
> MGET ingredient:fromage_molle ingredient:fruits_saison ingredient:farine ingredient:riz ingredient:pain ingredient:beurre ingredient:fromage_dure
ingredient:poire ingredient:huile_dolive ingredient:concombre ingredient:poulet ingredient:huile_M3_entree ingredient:bifteck
ingredient:pomme_de_terre ingredient:deux_oeufs ingredient:huile_dessert_M2 ingredient:huile_dolive_M1_M2 ingredient:yaourt ingredient:huile_M3_plat
ingredient:legumes_saisons ingredient:frite

1) "107"
2) "53.4"
3) "46.8"
4) "87.6"
5) "76"
6) "94.9"
7) "140"
8) "71"
9) "18.2"
10) "129"
11) "774"
12) "15.1"
13) "5370"
14) "15.5"
15) "276.7"
16) "32.3"
17) "18.5"
18) "360"
19) "17.1"
20) "53.4"
21) "260"
```

2. Stocker l'empreinte carbone de chaque plat et les ingredients associes :

Nous utilisons ici une clé Redis pour chaque plat et utilisez un type de données hash pour stocker les détails, par exemple :

HMSET plat:<nom> empreinte_carbone <empreinte_carbone> ingredient:<nom1> <empreinte_carbone1> ingredient:<nom2> < <empreinte_carbone> 2> …

Voici le résultat :

```
> HMSET plat:poulet_au_riz empreinte_carbone 1350 ingredient:poulet 774 ingredient:riz 84.6 ingredient:beurre 94.9

OK

> HMSET plat:bifteck_frite emprunte_carbone 5630 ingredient:bifteck 5370 ingredient:frite 260
```

```
> HMSET plat:omelette_pomme_de_terre empreinte_carbone 309.3 ingredient:deux_oeufs 276.7 ingredient:pomme_de_terre 15.5 ingredient:huile_M3_plat 17.1

OK

> HMSET entree:legumes_grecque empreinte_carbone 71.6 ingredient:legumes_saisons 53.4 ingredient:huile_dolive_M1_M2 18.2

OK

> HMSET entree:tzatziki empreinte_carbone 507.2 ingredient:yaourt 360 ingredient:concombre 129 ingredient:huile_dolive_M1_M2 18.2

OK
```

```
> HMSET entree:soupe_legume empreinte_carbone 68.5 ingredient:legumes_saisons 53.4 ingredient:huile_M3_entree 15.1

OK
```

```
> HMSET dessert:salade_fruit empreinte_carbone 129.4 ingredient:fruits_saison 53.4 ingredient:pain 76

OK

> HMSET dessert:tarte_poires empreinte_carbone 153.1 ingredient:farine 46.8 ingredient:poire 71 ingredient:huile_dessert_M2 32.3

OK

> HMSET dessert:plateau_fromage empreinte_carbone 323 ingredient:fromage_molle 107 ingredient:fromage_dure 140 ingredient:pain 76

OK
```

```
> TIME

1) 1684096630
2) 2123
```

3. Stocker la composition et l'empreinte carbone de chaque plat composant un repas :

Nous utilisons ici une clé Redis pour chaque menu et nous utilisons une liste pour stocker les plats qui le composent, suivant le modèle suivant :

LPUSH menu:<nom> plat:<plat1> plat:<plat2> …

```
> LPUSH repas:classique plat:poulet_au_riz entree:legumes_grecque dessert:plateau_fromage

(integer) 3
```

```
> LPUSH repas:vegeratien plat:omelette_pomme_de_terre entree:soupe_legume dessert:salade_fruit

(integer) 3
```

```
> LPUSH repas:classique_bis plat:bifteck_frite entree:tzatziki dessert:tarte_poires

(integer) 9
```

On vérifie si tout est bien là :

```
> KEYS repas*

1) "repas:classique"
2) "repas:classique_bis"
3) "repas:vegeratien"

> TIME

1) 1684097147
2) 740965
```

```
> LRANGE repas:classique 0 -1

1) "dessert:plateau_fromage"
2) "entree:legumes_grecque"
3) "plat:poulet_au_riz"
```

On répond donc à la question en affichant le temps d'exécution à la requête :

```
> HGETALL dessert:plateau_fromage

1) "empreinte_carbone"
2) "323"
3) "ingredient:fromage_molle"
4) "107"
5) "ingredient:fromage_dure"
6) "140"
7) "ingredient:pain"
8) "76"

> time

1) 1684097644
2) 416474
```

4. Trouver les plats contenant un ingrédient donné :

Nous utilisons ici une requête Redis pour rechercher les plats contenant un ingrédient spécifique, par exemple :

HGET <table>:<cle> <valeur>
HGETALL <table>:<cle>

```
> KEYS plat:*

1) "plat:bifteck_frite"
2) "plat:omelette_pomme_de_terre"
3) "plat:poulet_au_riz"

> HGET plat:poulet_au_riz empreinte_carbone

"1350"
```

```
> HGETALL plat:poulet_au_riz

1) "empreinte_carbone"
2) "1350"
3) "ingredient:poulet"
4) "774"
5) "ingredient:riz"
6) "84.6"
7) "ingredient:beurre"
8) "94.9"
```

Je n'ai pas relevé le temps, ni trouvé mieux pour répondre au mieux à la question.

5. Trouver les ingrédients, plats ou menus ayant la plus faible empreinte carbone ou une empreinte inférieure à un seuil donné :

Nous allons utiliser des requêtes Redis pour trier les ingrédients, les plats ou les menus en fonction de leur empreinte carbone, puis récupérez les éléments appropriés. Pour cela nous allons utiliser la notion d'ensemble (set) à l'aide des requête 'ZADD' qui va nous permettre de rentrer les clés associées à leur valeur empreinte carbone des hash que l'on a créé précédemment.
On créer pour cela une nouvelle table comme ci-dessous :

```
> ZADD empreinte_carbone 1350 empreinte_carbone:poulet_au_riz 5630 empreinte_carbone:bifteck_frite 309.3 empreinte_carbone:omelette_pomme_de_terre
(integer) 3
> ZRANGE empreinte_carbone 0 0 WITHSCORES
1) "empreinte_carbone:omelette_pomme_de_terre"
2) 309.3
```

```
> TIME

1) 1684100093
2) 695336
```

On répète cela pour chaque table des plats :

```
> ZADD empr_carb_dessert 323 empr_carb_dessert:plateau_fromage 153.1 empr_carb_dessert:tarte_poires 129.4 empr_carb_dessert:salade_fruit

(integer) 3

> ZRANGE empr_carb_dessert 0 0 WITHSCORES

1) "empr_carb_dessert:salade_fruit"
2) 129.4

> TIME

1) 1684100766
2) 995131
```

```
> ZADD empr_carb_entree 68.2 empr_carb_entree:soupe_legume 507.2 empr_carb_entree:tzatziki 71.6 empr_carb_entree:legume_grecque

(integer) 3

> ZRANGE empr_carb_entree 0 0 WITHSCORES

1) "empr_carb_entree:soupe_legume"
2) 68.2
```

```
> ZADD empr_carb_ingredient 107 empr_carb_ingredient:fromage_molle 53.4 empr_carb_ingredient:fruits_saison 46.8 empr_carb_ingredient:farine 87.6
empr_carb_ingredient:riz 76 empr_carb_ingredient:pain 94.9 empr_carb_ingredient:beurre 140 empr_carb_ingredient:fromage_dure 71
empr_carb_ingredient:poire 18.2 empr_carb_ingredient:huile_dolive 129 empr_carb_ingredient:concombre 774 empr_carb_ingredient:poulet 15.1
empr_carb_ingredient:huile_M3_entree 5370 empr_carb_ingredient:bifteck 15.5 empr_carb_ingredient:pomme_de_terre 276.7 empr_carb_ingredient:deux_oeufs
32.3 empr_carb_ingredient:huile_dessert_M2 18.5 empr_carb_ingredient:huile_dolive_M1_M2 360 empr_carb_ingredient:yaourt 17.1
empr_carb_ingredient:huile_M3_plat 53.4 empr_carb_ingredient:legumes_saisons 260 empr_carb_ingredient:frite

(integer) 21

> ZRANGE empr_carb_ingredient 0 0 WITHSCORES

1) "empr_carb_ingredient:huile_M3_entree"
2) 15.1

> TIME

1) 1684103682
2) 204031
```

```
> ZRANGEBYSCORE empr_carb_ingredient -inf 60

1) "empr_carb_ingredient:huile_M3_entree"
2) "empr_carb_ingredient:pomme_de_terre"
3) "empr_carb_ingredient:huile_M3_plat"
4) "empr_carb_ingredient:huile_dolive"
5) "empr_carb_ingredient:huile_dolive_M1_M2"
6) "empr_carb_ingredient:huile_dessert_M2"
7) "empr_carb_ingredient:farine"
8) "empr_carb_ingredient:fruits_saison"
9) "empr_carb_ingredient:legumes_saisons"

> TIME

1) 1684103929
2) 191540
```

Ces exemples fournissent une approche générale pour résoudre la problématique en utilisant Redis. Cependant, il est important de noter que Redis est principalement conçu comme une base de données en mémoire, et il peut être nécessaire de mettre en œuvre une logique supplémentaire en dehors de Redis pour gérer.

IV. MongoDB.

Nous allons présenter ici la façon dont on construit et on travaille une base de données avec la console MongoDB.

Par la suite nous expliciterons que les requêtes utilisées et leur résultat côté à côté.

Voici un exemple de construction d'une table dans le shell de MongoDB

```
db.ingredients.insertMany([
    { nom: 'legume_de_saison', empreinte_carbone: 53.4 },
    { nom: 'huile', empreinte_carbone: 18.2 },
    { nom: 'poulet', empreinte_carbone: 774 },
    { nom: 'riz', empreinte_carbone: 84.6 },
    { nom: 'beurre', empreinte_carbone: 94.9 },
    { nom: 'fromage_a_pate_molle', empreinte_carbone: 107 },
    { nom: 'fromage_a_pate_dure', empreinte_carbone: 140 },
    { nom: 'pain', empreinte_carbone: 76 },
    { nom: 'yaourt', empreinte_carbone: 360 },
    { nom: 'concombre', empreinte_carbone: 129 },
    { nom: 'bifteck', empreinte_carbone: 5370 },
    { nom: 'frite', empreinte_carbone: 260 },
    { nom: 'farine', empreinte_carbone: 46.9 },
    { nom: 'poire', empreinte_carbone: 71 },
    { nom: 'huile_cuilleree', empreinte_carbone: 32.3 },
    { nom: 'huile_demi_cuilleree', empreinte_carbone: 15.1 },
    { nom: 'deux_oeufs', empreinte_carbone: 276.7 },
    { nom: 'pomme_de_terre', empreinte_carbone: 15.5 },
    { nom: 'huile_demi_cuil_vegan', empreinte_carbone: 17.1 },
    { nom: 'fruits_saison', empreinte_carbone: 53.4 }
]);
```

```
db.plat.insertMany([
    {
        nom: "bifteck-frite",
        ingredients: [
            { nom: "bifteck", empreinte_carbone: 5370 },
            { nom: "frite", empreinte_carbone: 260 }
        ],
        empreinte_carbone:5630
    },
    {
        nom: "poulet au riz",
        ingredients: [
            { nom: "poulet", empreinte_carbone: 774 },
            { nom: "riz", empreinte_carbone: 84.6 },
            { nom: "beurre", empreinte_carbone: 94.9 }
        ],
        empreinte_carbone:953.5
    },
    {
        nom: "omelette aux pomme de terre",
        ingredients: [
            { nom: "pomme_de_terre", empreinte_carbone: 276.7 },
            { nom: "deux_oeufs", empreinte_carbone: 15.5 },
            { nom: "huile_demi_cuil_vegan", empreinte_carbone: 17.1 }
        ],
        empreinte_carbone:309.3
    }
```

1. Question 1 :

```
// Question 1 : Recherche de l'empreinte carbone de l'ingrédient "riz"
db.ingredients.findOne({ nom: "riz" }).empreinte_carbone;
```

```
}
84.6
```

2. Question 2 :

```
const plat = db.plat.findOne({ nom: "poulet au riz" });
const empreinteCarbone = plat.empreinte_carbone;
const ingredients = plat.ingredients.map(ingredients => {
    const ingredientDetails = db.ingredients.findOne({ nom: ingredients.nom
    return {
        nom: ingredientDetails.nom,
        empreinte_carbone: ingredients.empreinte_carbone
    };
});

// Affichage des résultats
print("Empreinte carbone du plat : " + empreinteCarbone);
print("Ingrédients du plat :");
ingredients.forEach(ingredients => {
    print("- " + ingredients.nom + " : " + ingredients.empreinte_carbone);
});
```

NEW MONGODB RUN

STDIN

Input for the program (Optional)

```
Empreinte carbone du plat : 953.5
Ingrédients du plat :
- poulet : 774
- riz : 84.6
- beurre : 94.9
```

3. Question 3

```
const repas = db.repas.findOne({ nom: "repas classique" });

repas.plat.ingredients.forEach(ingredients => {
    const ingredientDetails_2 = db.ingredients.findOne({ nom: ingredients.nom });
    ingredients.empreinte_carbone = ingredientDetails_2.empreinte_carbone;
});

// Affichage des résultats
print("Composition et empreinte carbone des plats :");
repas.plat.ingredients.forEach(ingredients => {
    print("- " + ingredients.nom + " : " + ingredients.empreinte_carbone);
});

print("Empreinte carbone totale du repas : " + repas.total_carbone);
```

NEW MONGODB RUN

STDIN

Input for the program (Optional)

```
Composition et empreinte carbone des plats :
- poulet : 774
- riz : 84.6
- beurre : 94.9
Empreinte carbone totale du repas : 1350
```

4. Question 4

```
const ingredient = "riz";
const plats = db.plat.find({ "ingredients.nom" : ingredient },
{ nom: 1, empreinte_carbone: 1 });

// Affichage des résultats
print("Plats contenant l'ingrédient '" + ingredient + "' :");
plats.forEach(plat => {
    print("- " + plat.nom + " : " + plat.empreinte_carbone);
});
```

```
Plats contenant l'ingrédient 'riz' :
- poulet au riz : 953.5
```

5. Question 5

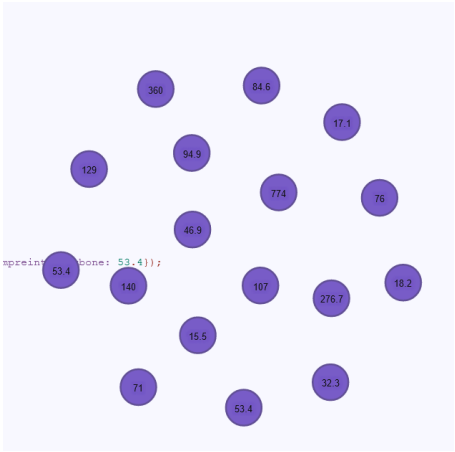
```
const seuil1 = 50;
const seuil2 = 500;
const seuil3 = 550;
// Recherche des ingrédients
const ingredients_2 = db.ingredients.find({ empreinte_carbone: { $lt: seuil1 } },
{ nom: 1, empreinte_carbone: 1 });
// Recherche des plats
const plats_2 = db.plat.find({ empreinte_carbone: { $lt: seuil2 } },
{ nom: 1, empreinte_carbone: 1 });
// Recherche des menus
const menus = db.repas.find({ total_carbone: { $lt: seuil3 } },
{ nom: 1, total_carbone: 1 });
// Affichage des résultats
print("Ingrédients ayant une empreinte carbone inférieure à " + seuil1 + " :");
ingredients_2.forEach(ingredients => {
    print("- " + ingredients.nom + " : " + ingredients.empreinte_carbone);
});

print("Plats ayant une empreinte carbone inférieure à " + seuil2 + " :");
plats_2.forEach(plat => {
    print("- " + plat.nom + " : " + plat.empreinte_carbone);
});

print("Menus ayant une empreinte carbone inférieure à " + seuil3 + " :");
menus.forEach(menu => {
    print("- " + menu.nom + " : " + menu.total_carbone);
});
```

```
Ingrédients ayant une empreinte carbone infé
- huile : 18.2
- farine : 46.9
- huile_cuilleree : 32.3
- huile_demi_cuilleree : 15.1
- pomme_de_terre : 15.5
- huile_demi_cuil_vegan : 17.1
Plats ayant une empreinte carbone inférieure
- omelette aux pomme de terre : 309.3
Menus ayant une empreinte carbone inférieure
- repas végétarien : 510
```

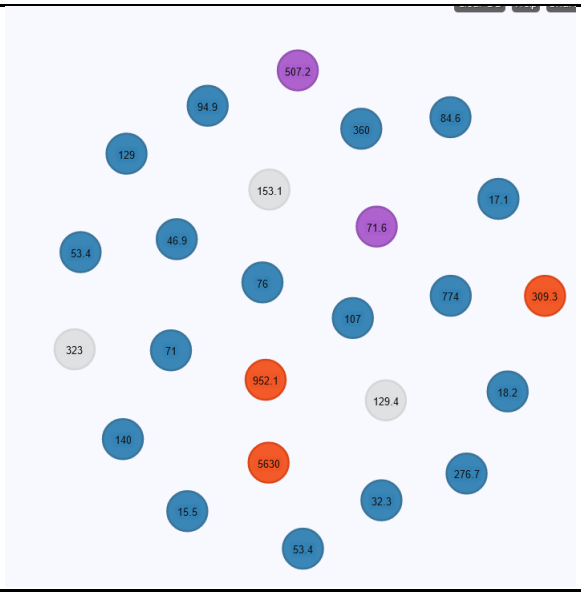
V. Neo4j

Nous allons présenter pour la forme de la construction de la base de données. Dans le rendu final du projet, les requêtes sous Neo4j sont un peu longues et peu explicites.

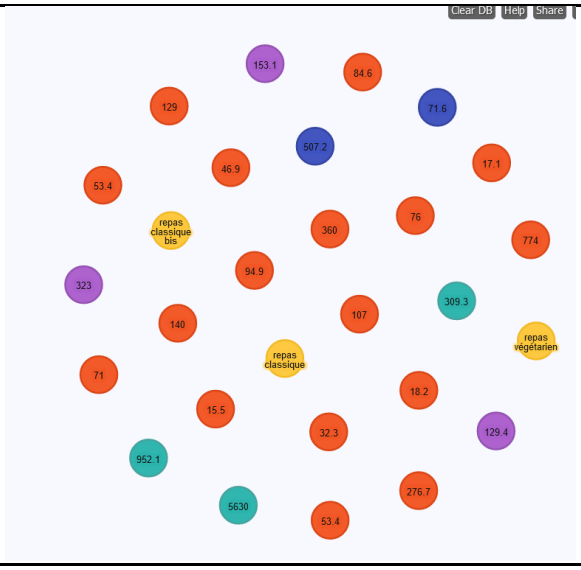
```
CREATE (lde:Ingredient {nom: 'legume_de_saison',
empreinte_carbone: 53.4})
CREATE (hdo:Ingredient {nom: 'huile_d_olive',
empreinte_carbone: 18.2})
CREATE (pou:Ingredient {nom: 'poulet', empreinte_carbone:
774})
CREATE (riz:Ingredient {nom: 'riz', empreinte_carbone:
84.6})
CREATE (beu:Ingredient {nom: 'beurre', empreinte_carbone:
94.9})
CREATE (fpm:Ingredient {nom: 'fromage_a_pate_molle',
empreinte_carbone: 107})
CREATE (fpd:Ingredient {nom: 'fromage_a_pate_dure',
empreinte_carbone: 140})
CREATE (pai:Ingredient {nom: 'pain', empreinte_carbone: 76})
CREATE (yao:Ingredient {nom: 'yaourt', empreinte_carbone:
360})
CREATE (con:Ingredient {nom: 'concombre', empreinte_carbone:
129})
CREATE (bif:Ingredient {nom: 'bifteck', empreinte_carbone:
5370})
CREATE (fri:Ingredient {nom: 'frite', empreinte_carbone:
260})
CREATE (far:Ingredient {nom: 'farine', empreinte_carbone:
46.9})
CREATE (poi:Ingredient {nom: 'poire', empreinte_carbone:
71})
CREATE (huo:Ingredient {nom: 'huile', empreinte_carbone:
32.3})
CREATE (deu:Ingredient {nom: 'deux_oeufs',
empreinte_carbone: 276.7})
CREATE (pom:Ingredient {nom: 'pomme_de_terre',
empreinte_carbone: 15.5})
CREATE (fra:Ingredient {nom: 'fruits_de_saison',
empreinte_carbone: 53.4});
```



```
CREATE (ome:Plat {nom: 'omelette_pomme_de_terre',
empreinte_carbone: 309.3})
CREATE (pou:Plat {nom: 'poulet_au_riz', empreinte_carbone:
952.1})
CREATE (bif:Plat {nom: 'bifteck_frite', empreinte_carbone: 5630});
CREATE (lgg:Entree {nom: 'legumes_a_la_grecque',
empreinte_carbone: 71.6})
CREATE (tzt:Entree {nom: 'tzatziki', empreinte_carbone: 507.2})
CREATE (sou:Entree {nom: 'soupe_de_legume',
empreinte_carbone: 68.5});

CREATE (pdf:Dessert {nom: 'plateau_de_fromage',
empreinte_carbone: 323})
CREATE (sdf:Dessert {nom: 'salade_de_fruit', empreinte_carbone:
129.4})
CREATE (tpr:Dessert {nom: 'tarte_poire', empreinte_carbone:
153.1});
```

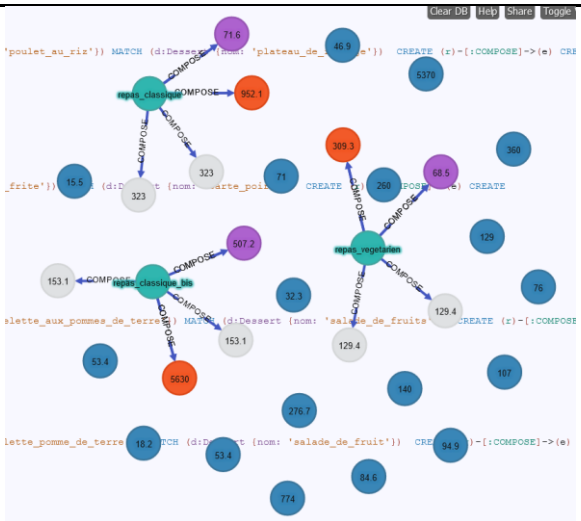



```
CREATE (rc:Repas {nom: 'repas_classique',
total_empreinte_carbone: 1350})
CREATE (rcb:Repas {nom: 'repas_classique_bis',
total_empreinte_carbone: 6290})
CREATE (rv:Repas {nom: 'repas_vegetarien',
total_empreinte_carbone: 510});
```



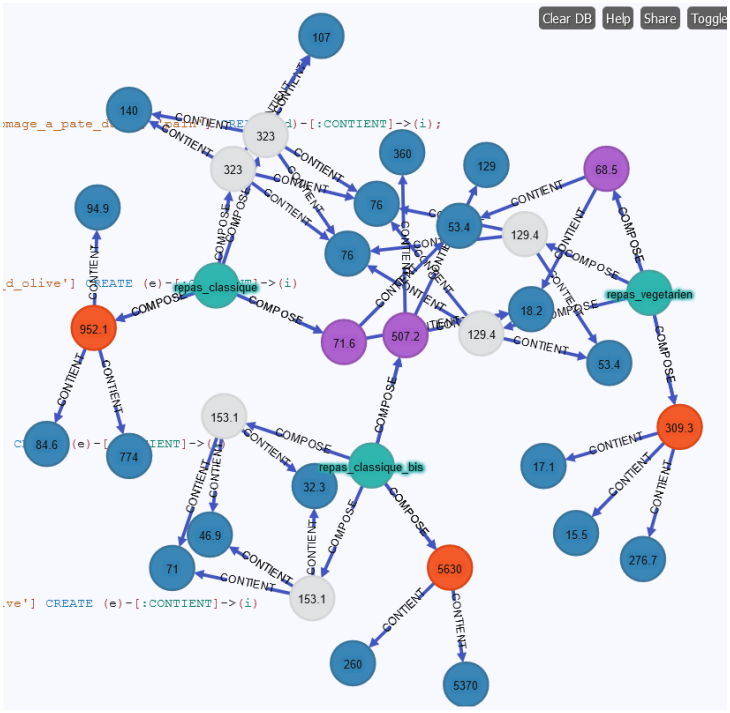
Création de l'équivalent des tables de liaison en Neo4j :

```
MATCH (r:Repas {nom: 'repas_classique'})
MATCH (e:Entree {nom: 'legumes_a_la_grecque'})
MATCH (p:Plat {nom: 'poulet_au_riz'})
MATCH (d:Dessert {nom: 'plateau_de_fromage'})
CREATE (r)-[:COMPOSE]->(e)
CREATE (r)-[:COMPOSE]->(p)
CREATE (r)-[:COMPOSE]->(d);


MATCH (p:Plat), (i:Ingredient)
WHERE p.nom = 'omelette_pomme_de_terre' AND i.nom IN
['deux_oeufs', 'pomme_de_terre', 'huile_demi_cuil_vegan']
CREATE (p)-[:CONTIENT]->(i);

MATCH (p:Plat), (i:Ingredient)
WHERE p.nom = 'poulet_au_riz' AND i.nom IN ['poulet', 'riz',
'beurre']
CREATE (p)-[:CONTIENT]->(i);
```



MATCH (p:Plat), (i:Ingredient) WHERE p.nom = 'bifteck_frite' AND i.nom IN ['bifteck', 'frite'] CREATE (p)-[:CONTIENT]->(i);	

Etat finale de la base de donne avec le graphe après quelque remodelage sur les tables :



Voilà maintenant nos bases de données ainsi créer. Nous allons répondre aux questions demandées.

1. Question 1 :

Query:
MATCH (i:Ingredient {nom: 'riz'})
RETURN i.empreinte_carbone;

i.empreinte_carbone
84.6

Query took 6 ms and returned 1 rows.

2. Question 2 :

Query:
MATCH (p:Plat {nom: 'poulet_au_riz'})-[:CONTIENT]->(i:Ingredient)
RETURN p.nom AS nom_plat, COLLECT(i.nom) AS ingredients, COLLECT(i.empreinte_carbone) AS empreinte_carbone_ingredients;

nom_plat ingredients empreinte_carbone_ingredients
poulet_au_riz [riz, beurre, poulet] [84.6, 94.9, 774]

Query took 16 ms and returned 1 rows.

Query:
MATCH (p:Plat {nom: 'omelette_pomme_de_terre'})-[:CONTIENT]->(i:Ingredient)
RETURN p.nom AS nom_plat, COLLECT(i.nom) AS ingredients, COLLECT(i.empreinte_carbone) AS empreinte_carbone_ingredients;

nom_plat ingredients empreinte_carbone_ingredients
omelette_pomme_de_terre [pomme_de_terre, deux_oeufs, huile_demi_cuil_vegan] [15.5, 276.7, 17.1]

Query took 10 ms and returned 1 rows.

3. Question 3 :

Query:
MATCH (m:Repas)-[:COMPOSE]->(p:Plat) OPTIONAL
MATCH (p)-[:CONTIENT]->(i:Ingredient)
RETURN m.nom AS nom_menu, p.nom AS nom_plat, p.empreinte_carbone AS empreinte_carbone_plat, COLLECT(i.nom) AS ingredients, COLLECT(i.empreinte_carbone) AS empreinte_carbone_ingredients;

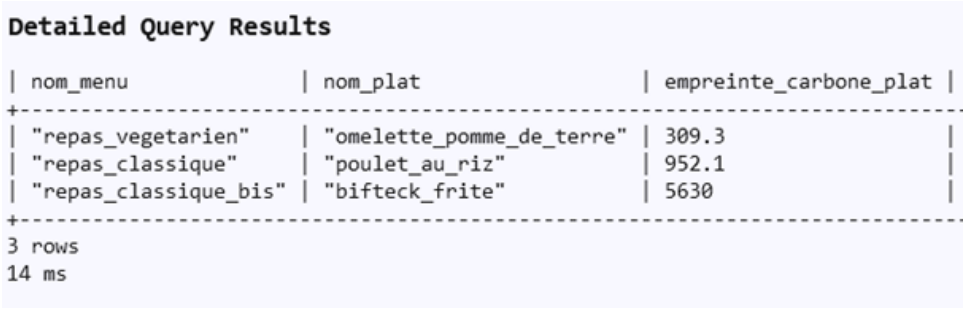
Query took 13 ms and returned no rows.

Query:
MATCH (m:Repas {nom : 'repas_classique'})-[:COMPOSE]->(p:Plat) OPTIONAL

MATCH (p)-[:CONTIENT]->(i:Ingredient) RETURN m.nom AS nom_menu, p.nom AS nom_plat, p.empreinte_carbone AS empreinte_carbone_plat, COLLECT(i.nom) AS ingredients, COLLECT(i.empreinte_carbone) AS empreinte_carbone_ingredients;

nom_menu nom_plat empreinte_carbone_plat ingredients empreinte_carbone_ingredients
repas_classiquepoulet_au_riz 952.1 [riz, beurre, poulet, riz, beurre, poulet] [84.6, 94.9, 774, 84.6, 94.9, 774]

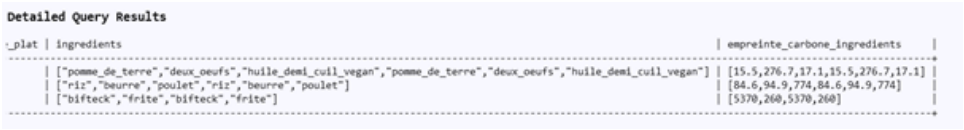
Query took 24 ms and returned 1 rows.



Detailed Query Results

nom_menu	nom_plat	empreinte_carbone_plat
"repas_vegetarien"	"omelette_pomme_de_terre"	309.3
"repas_classique"	"poulet_au_riz"	952.1
"repas_classique_bis"	"bifteck_frite"	5630

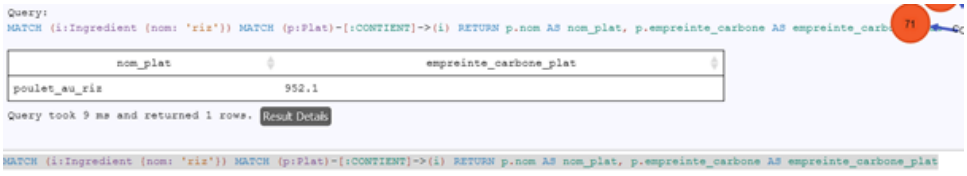
3 rows
14 ms



Detailed Query Results

_plat	ingredients	empreinte_carbone_ingredients
	["pomme_de_terre","deux_oeufs","huile_demi_cuil_vegan","pomme_de_terre","deux_oeufs","huile_demi_cuil_vegan"]	[15.5,276.7,17.1,15.5,276.7,17.1]
	["riz","beurre","poulet","riz","beurre","poulet"]	[84.6,94.9,774,84.6,94.9,774]
	["bifteck","frite","bifteck","frite"]	[5370,260,5370,260]

4. Question 4 :



Query:
MATCH (i:Ingredient {nom: 'riz'}) MATCH (p:Plat)-[:CONTIENT]->(i) RETURN p.nom AS nom_plat, p.empreinte_carbone AS empreinte_carbone_plat

nom_plat	empreinte_carbone_plat
poulet_au_riz	952.1

Query took 9 ms and returned 1 rows. Result Details

MATCH (i:Ingredient {nom: 'riz'}) MATCH (p:Plat)-[:CONTIENT]->(i) RETURN p.nom AS nom_plat, p.empreinte_carbone AS empreinte_carbone_plat

Query:
MATCH (i:Ingredient {nom: 'riz'}) MATCH (p:Plat)-[:CONTIENT]->(i) RETURN p.nom AS nom_plat, p.empreinte_carbone AS empreinte_carbone_plat

nom_plat empreinte_carbone_plat
poulet_au_riz 952.1

Query took 9 ms and returned 1 rows.

Query:

MATCH (m:Repas)-[:COMPOSE]->(p:Plat) OPTIONAL MATCH (p)-[:CONTIENT]->(i:Ingredient) RETURN m.nom AS nom_menu, p.nom AS nom_plat, p.empreinte_carbone AS empreinte_carbone_plat, COLLECT(i.nom) AS ingredients, COLLECT(i.empreinte_carbone) AS empreinte_carbone_ingredients;

nom_menu nom_plat empreinte_carbone_plat ingredients empreinte_carbone_ingredients
repas_vegetarien omelette_pomme_de_terre 309.3 [pomme_de_terre, deux_oeufs, huile_demi_cuil_vegan, pomme_de_terre, deux_oeufs, huile_demi_cuil_vegan] [15.5, 276.7, 17.1, 15.5, 276.7, 17.1]
repas_classiquepoulet_au_riz 952.1 [riz, beurre, poulet, riz, beurre, poulet] [84.6, 94.9, 774, 84.6, 94.9, 774]
repas_classique_bis bifteck_frite 5630 [bifteck, frite, bifteck, frite] [5370, 260, 5370, 260]

Query took 14 ms and returned 3 rows.

5. Question 5 :

Query:
MATCH (i:Ingredient)
WHERE i.empreinte_carbone <= 50 WITH COLLECT(i) AS ingredients
MATCH (p:Plat) WHERE p.empreinte_carbone <= 500 WITH ingredients, COLLECT(p) AS plats
MATCH (m:Repas) WHERE m.total_empreinte_carbone <= 550 WITH ingredients, plats, COLLECT(m) AS menus
RETURN ingredients, plats, menus;

ingredients plats menus
[(1:Ingredient {empreinte_carbone:18.2, nom:"huile_d_olive"}), (20:Ingredient {empreinte_carbone:46.9, nom:"farine"}), (22:Ingredient {empreinte_carbone:32.3, nom:"huile"}), (24:Ingredient {empreinte_carbone:15.5, nom:"pomme_de_terre"}), (123:Ingredient {empreinte_carbone:17.1, nom:"huile_demi_cuil_vegan"})] [(28:Plat {empreinte_carbone:309.3, nom:"omelette_pomme_de_terre"})] [(106:Repas {nom:"repas_vegetarien", total_empreinte_carbone:510})]

On remarque ici que l'exécution de la dernière requête prend 30ms. Soi plus de temps que sous PostgresSQL dans les deux modèles que nous avons utilisés. Par ailleurs, la création de colonne et de leur taille pour répondre à la question est plus réduite. Ainsi en plus grande donnée Neo4J est plus efficace qu'en utilisation du SQL simple.

VI. Conclusion

Je n’ai malheureusement pas pu faire rentrer dans ce rapport l’étude sur l’analyse à plus grande échelle les modèles. J’ai voulu rester succincte et présenter la conclusion sur les performances de chaque modèle du mieux que j’ai pu.

J’ia beaucoup aimé travailler avec Neo4j. Même si la prise en main prend du temps, son utilisation reste très modulable et la représentation graphique nous guide tout au long des requêtes.

Par ailleurs, une difficulté fut avec Redis. J'ai peut-être mal compris ce que l’on a fait en TD et ce que j’ai retravailler avec les tutos chez moi, je n’ai pas spécifiquement très adapté Redis aux questions demandées.

Enfin, pour PostgreSQL et MongoDB, la difficulté fut assez mitigée. S’il n’y a pas beaucoup de référence sur internet qui nous guide convenablement pour bien comprendre à comment implémenter les requêtes, on ne s’en sort pas. Même si la requête SQL ou le format JSON reste assez accessible, les demandes du projet ont demandé un approfondissement des notions de ces formats pour mieux entreprendre la réponse aux questions.