UQÀM | Faculté des sciences
Université du Québec à Montréal

INF8810 – **Traitement et analyse des données massives**

Travail en groupe de 4 personnes

Projet de Session #2 :

CRÉATION D'UN SYSTÈME DE RECOMMANDATION AVEC NEO4J

Session d'Automne 2022

Partie 1 : chargement des données dans neo4j (nœuds, relations et propriétés)

Description des étapes pour le prétraitement et le chargement des données

- **Étape 1 : Copier les fichiers dans le répertoire d'importation par défaut de Neo4j**
  Nous copions nos fichiers de l'environnement local dans le répertoire d'importation de Neo4j et accordons toutes les permissions pour que les fichiers soient accessibles par tous les utilisateurs.

```
magariou@openonevm:~/Downloads/Projet_2/Datasets$ sudo cp * /var/lib/neo4j/import/
[sudo] password for magariou:
magariou@openonevm:~/Downloads/Projet_2/Datasets$ sudo chmod 777 /var/lib/neo4j/import/
magariou@openonevm:~/Downloads/Projet_2/Datasets$ ls -la /var/lib/neo4j/import/
total 367368
drwxrwxrwx 2 neo4j adm       4096 Nov 30 02:05 .
drwxr-xr-x 8 neo4j adm       4096 Nov 17 20:35 ..
-rwxrwxrwx 1 root  root    163557 Nov 25 04:13 movies.csv
-rwxrwxrwx 1 root  root  21593504 Nov 25 04:13 ratings.csv
-rwxrwxrwx 1 root  root      5577 Nov 25 04:13 README
-rw-r--r-- 1 root  root  79487367 Nov 30 02:05 spotify_albums.csv
-rw-r--r-- 1 root  root   6520524 Nov 30 02:05 spotify_artists.csv
-rw-r--r-- 1 root  root 268277198 Nov 30 02:05 spotify_tracks.csv
-rwxrwxrwx 1 root  root    110208 Nov 25 04:13 users.csv
magariou@openonevm:~/Downloads/Projet_2/Datasets$ sudo chmod 777 /var/lib/neo4j/import/*
magariou@openonevm:~/Downloads/Projet_2/Datasets$ ls -la /var/lib/neo4j/import/
total 367368
drwxrwxrwx 2 neo4j adm       4096 Nov 30 02:05 .
drwxr-xr-x 8 neo4j adm       4096 Nov 17 20:35 ..
-rwxrwxrwx 1 root  root    163557 Nov 25 04:13 movies.csv
-rwxrwxrwx 1 root  root  21593504 Nov 25 04:13 ratings.csv
-rwxrwxrwx 1 root  root      5577 Nov 25 04:13 README
-rwxrwxrwx 1 root  root  79487367 Nov 30 02:05 spotify_albums.csv
-rwxrwxrwx 1 root  root   6520524 Nov 30 02:05 spotify_artists.csv
-rwxrwxrwx 1 root  root 268277198 Nov 30 02:05 spotify_tracks.csv
-rwxrwxrwx 1 root  root    110208 Nov 25 04:13 users.csv
```

- **Étape 2 : Suppression de tous les nœuds préexistants dans Neo4j.**
  Dans cette étape, nous vidons complètement la base de données Neo4j avant de charger nos données pour éviter toute ambiguïté.

```
neo4j@neo4j> match (n) detach delete n;
0 rows available after 342 ms, consumed after another 0 ms
Deleted 9969 nodes, Deleted 18488 relationships
neo4j@neo4j>
```

- **Étape 3 : Identification des colonnes (propriétés) pertinentes pour chaque fichier de notre projet.**
  Pour la création de notre requête de recommandation, nous avons sélectionné certaines colonnes (propriétés) dans chaque fichier que nous avons jugées pertinentes pour la création du système de recommandation.

Pour le fichier spotify_albums :

album_type,artist_id,available_markets,id,name,release_date,total_tracks,track_id,type

Pour le fichier spotify_artists :

artist_popularity,followers,genres,id,name,track_id, type

Pour le fichier spotifiy_tracks :

acousticness,album_id,artists_id,available_markets,country,danceability,disc_number,duration_ms,energy,id, instrumentalness,key,liveness,loudness,mode,name,playlist, popularity, speechiness,tempo,track_number,valence, type

- **Étape 4 : Chargement et prétraitement des données**

  Note : pour cette étape, nous avons effectué notre prétraitement lors du chargement des données (ex. : conversion du type de données, gestion des listes).

  Nous avons utilisé un échantillon des données pour pouvoir générer des résultats rapidement afin de tester notre modèle.

  ***Création du nœud Albums* :**

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM 'file:///spotify_albums.csv' as row
MERGE (a:Albums {album_type: row.album_type,artist_id: row.artist_id,id:
row.id,name: row.name,release_date: row.release_date,
total_tracks: toInteger(row.total_tracks),track_id: row.track_id,type: row.type})
WITH a, row
UNWIND split(row.available_markets, ',') as available_markets
MERGE (av_m:Available {nom: available_markets})
MERGE (a)-[r:disponible]->(av_m);
```

  Note : *comme la disponibilité de l'album sur les marchés est une liste d'éléments, nous l'avons spliter en colonne de valeur et avons créé une relation du nœud Available avec le nœud Albums.*

  Création du nœud Artists :

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM 'file:///spotify_artists.csv' as row
MERGE (art:Artists {artist_popularity: toFloat(row.artist_popularity),followers:
toFloat(row.followers),id: row.id,name: row.name,track_id: row.track_id,type: row.type})
with art, row, replace(row.genres,'[','') as genres_et1
with art, row, replace(genres_et1,']','') as genres_et2
with art, row, replace(genres_et2,"'","") as genres_et3
with art, row, genres_et3 UNWIND split(genres_et3,',') as genres
MERGE (g:Genre {nom: genres})
MERGE (art)-[r:a_le_genre]->(g);
```

***Note** : le genre était considéré comme une chaine de caractères dans le fichier, alors qu'il s'agissait d'une liste d'éléments. Donc, nous avons effectué un prétraitement sur cette colonne (propriété) afin de la transformer en une liste et transformer cette liste en colonne. Finalement, nous avons créé une relation entre le nœud Genre et Artists.*

Création du nœud Tracks :

```
USING PERIODIC COMMIT 500
    LOAD CSV WITH HEADERS FROM 'file:///spotify_tracks.csv' as row
    WITH row
    WHERE row.country is not null and row.disc_number is not null
     with row, replace(row.artists_id,'[','') as artist_et1
     with row, replace(artist_et1,']','') as artist_et2
     with row, replace(artist_et2,"'",'') as artist_et3
    with row, split(artist_et3,',') as artist_ids
    CREATE (t:Tracks {
    acousticness: toFloat(row.acousticness),
    album_id: row.album_id,
    artists_id: artist_ids,
    country: row.country,
    danceability: toFloat(row.danceability),
    disc_number: toFloat(row.disc_number),
    duration_ms: toFloat(row.duration_ms),
    energy: toFloat(row.energy),
    id: row.id,
    instrumentalness: toFloat(row.instrumentalness),
    key: toFloat(row.key),
    liveness: toFloat(row.liveness),
    loudness: toFloat(row.loudness),
    mode: toFloat(row.mode),
    name: row.name,
    playlist: row.playlist,
    popularity: toFloat(row.popularity),
    speechiness: toFloat(row.speechiness),
    tempo: toFloat(row.tempo),
    track_number: toFloat(row.track_number),
    valence: toFloat(row.valence),
    type: row.type})
    WITH t, row
    UNWIND split(row.available_markets, ',') as available_markets
    MERGE (av:Available_Markets {nom: available_markets})
    MERGE (t)-[r:est_disponible_sur]->(av);
```

***Note** : les identifiants des artistes qui ont participé sur une chanson étaient considérés comme une chaine de caractères dans le fichier, alors qu'il s'agissait d'une liste d'éléments. Donc, nous avons effectué un prétraitement sur cette colonne afin de la transformer en une liste des valeurs (identifiants des artistes). Finalement, nous avons transformé les valeurs de liste de la disponibilité de la chanson en colonne et crée une relation entre le nœud Available_Markets et Tracks.*

- **Étape 5 : Création des relations entre les trois nœuds**
  Nous avons jugé de faire ces relations à cette étape pour ne pas trop surcharger les requêtes de l'étape précédente. Nous avons créé des relations entre les nœuds Artists, Tracks et Albums.

Relation entre Artists et Tracks

```
MATCH (art:Artists)
MATCH (t:Tracks)
WHERE  art.id IN t.artists_id
MERGE (art)-[r:A_chante]->(t);
```

Un aperçu

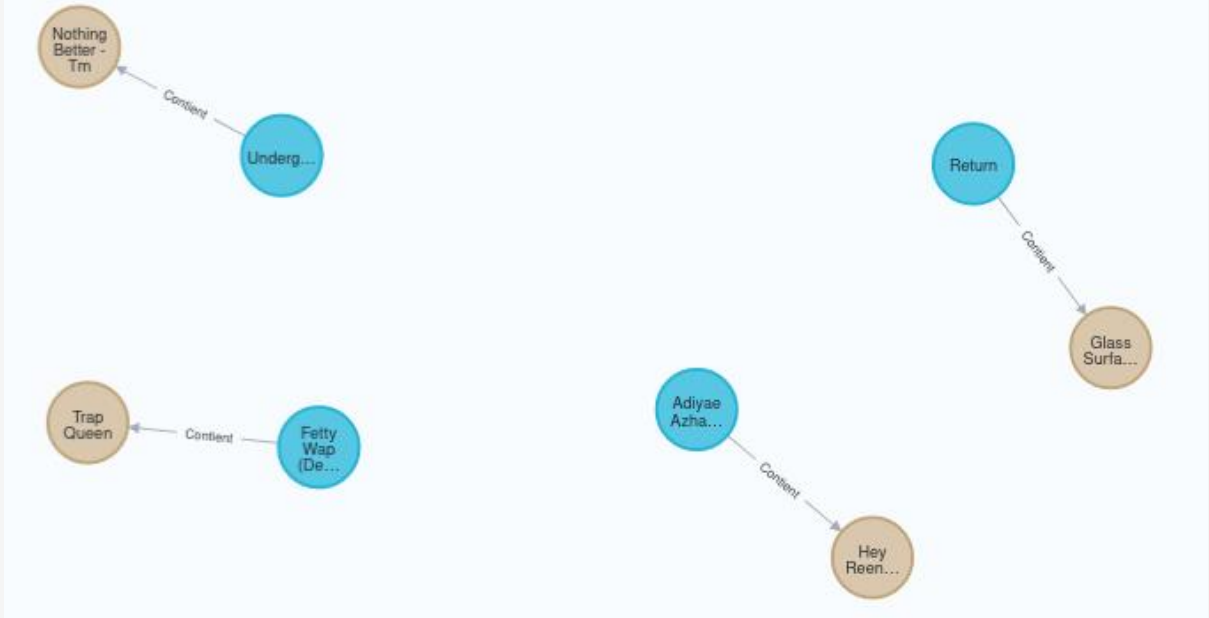
```
match(ar:Artists)-[r]->(tr:Tracks) return ar, r, tr;
```

Relation entre Albums et Tracks

```
MATCH (al:Albums)
MATCH (t:Tracks)
WHERE  al.id = t.album_id
MERGE (al)-[r:Contient]->(t);
```

Un aperçu

```
match(al:Albums)-[r]->(tr:Tracks) return al , r, tr;
```

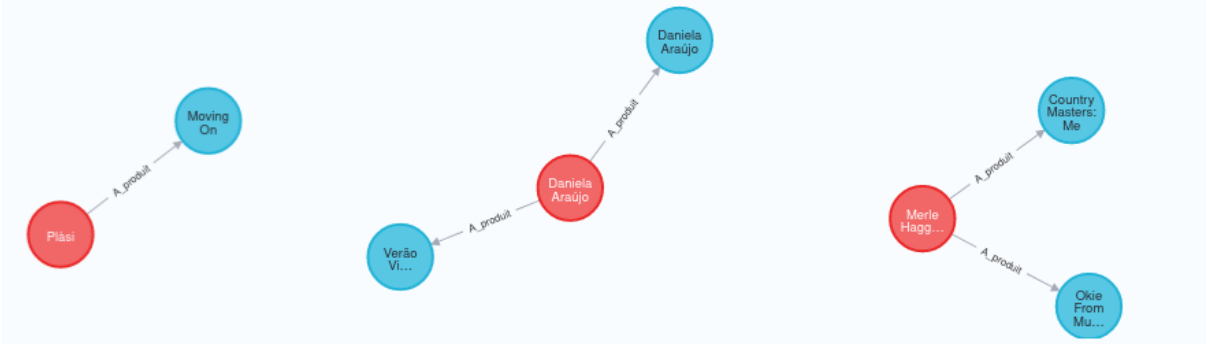


Relation entre Artists et Albums

```
MATCH (al:Albums)
MATCH (art:Artists)
WHERE al.artist_id = art.id
MERGE (art)-[r:A_produit]->(al);
```

Un aperçu

```
match(ar:Artists)-[r]->(al:Albums) return ar,r, al;
```



Partie 2 : Création d'une requête de recommandation

Dans le cadre de l'élaboration d'un système de recommandation, nous avons choisi d'utiliser deux approches différentes :

- Premièrement, nous avons utilisé la **distance euclidienne** en prenant en compte toutes les propriétés d'une chanson en fonction d'une chanson donnée en entrée. La formule de calcul est la suivante :

$$distance(I, C) = \sqrt{(x_C - x_I)^2 + (y_C - y_I)^2}$$

Ensuite, nous avons calculé la distance entre chaque propriété de la chanson en entré, par rapport à toutes les autres chansons. Le résultat de la distance de chaque propriété est stocké l'intérieur d'une relation (porteuse d'information).

- Deuxièmement, nous avons utilisé le **cosinus de similarité** pour donner des poids à nos relations afin de déterminer les chansons qui ont les similarités les plus élevées pour les recommander à l'utilisateur.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Après une analyse des données, nous avons remarqué une grande différence de grandeur entre les valeurs. Nous avons opté pour le **minmaxscaler** pour **normaliser** les données :

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Requête de la recommandation à partir de la distance euclidienne

1
```
match(TT:Tracks) with min(TT.acousticness) as min_ac, max(TT.acousticness) as max_ac,
min(TT.danceability) as min_dc, max(TT.danceability) as max_dc,
min(TT.energy) as min_en, max(TT.energy) as max_en,
min(TT.instrumentalness) as min_ins, max(TT.instrumentalness) as max_ins,
min(TT.liveness) as min_li, max(TT.liveness) as max_li,
min(TT.loudness) as min_lo, max(TT.loudness) as max_lo,
min(TT.speechiness) as min_sp, max(TT.speechiness) as max_sp,
min(TT.tempo) as min_tp, max(TT.tempo) as max_tp,
min(TT.valence) as min_va, max(TT.valence) as max_va
```

2
```
match(T:Tracks) with T, min_ac, max_ac, min_dc, max_dc, min_en, max_en, min_ins,
max_ins, min_li, max_li, min_lo, max_lo, min_sp, max_sp, min_tp, max_tp, min_va,
max_va where T.id = "3aEEfw8PNGpqdWFKOMPeJN"
```

3
```
match(C:Tracks) with T, C, min_ac, max_ac, min_dc, max_dc, min_en, max_en, min_ins,
max_ins, min_li, max_li, min_lo, max_lo, min_sp, max_sp, min_tp, max_tp, min_va,
max_va where C.id <> "3aEEfw8PNGpqdWFKOMPeJN"
```

4

```
merge (T)-[r:distance_euc_1{euc: sqrt(
((T.acousticness-min_ac)/(max_ac-min_ac)-(C.acousticness-min_ac)/(max_ac-min_ac))^2+
((T.danceability-min_dc)/(max_dc-min_dc)-(C.danceability-min_dc)/(max_dc-min_dc))^2+
((T.energy-min_en)/(max_en-min_en)-(C.energy-min_en)/(max_en-min_en))^2+
((T.instrumentalness-min_ins)/(max_ins-min_ins)-(C.instrumentalness-min_ins)/(max_ins-
min_ins))^2+
 ((T.liveness-min_li)/(max_li-min_li)-(C.liveness-min_li)/(max_li-min_li))^2+
((T.loudness-min_lo)/(max_lo-min_lo)-(C.loudness-min_lo)/(max_lo-min_lo))^2+
((T.speechiness-min_sp)/(max_sp-min_sp)-(C.speechiness-min_sp)/(max_sp-min_sp))^2+
((T.tempo-min_tp)/(max_tp-min_tp)-(C.tempo-min_tp)/(max_tp-min_tp))^2+
((T.valence-min_va)/(max_va-min_va)-(C.valence-min_va)/(max_va-min_va))^2
)}]-> (C)
```

5

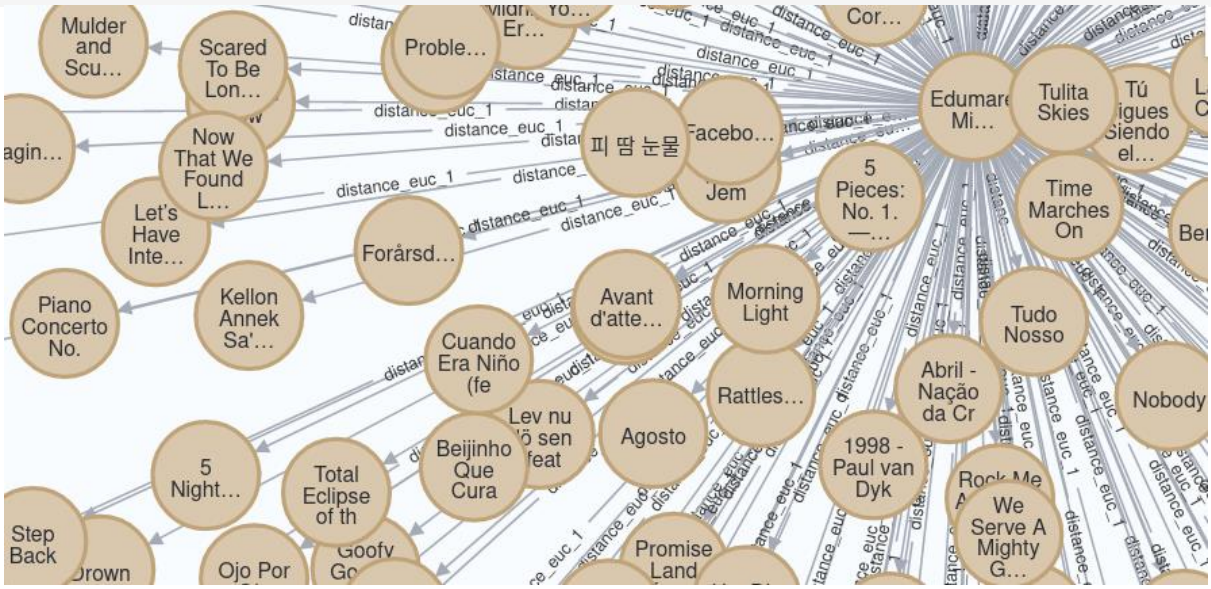
```
with T,r,C order by r.euc limit 3
match (m:Tracks {id: C.id})<-[]-(A:Artists)
match (Trk:Tracks {id: C.id})<-[]-(Al:Albums )

return Al.name, C.name, COLLECT (A.name) as Artist_name, C.country, C.type, r.euc;
```

Résultat

Liste des chansons en fonction de la distance euclidienne

Al.name	C.name	Artist_name	C.country	C.type	r.euc
"80-luvun suursuosikit"	"Tahdon sinut"	["Meiju Suvas"]	"FI"	"track"	0.14872292310963053
"Moonshine Freeze"	"Moonshine Freeze"	["This Is The Kit"]	"FI"	"track"	0.19548331493314403
"Spirit in the Dark"	"Spirit in the Dark"	["Aretha Franklin"]	"FI"	"track"	0.20113971660146654

```
match(tr:Tracks)-[r:distance_euc_1]->(trs:Tracks) return tr, r, trs;
```



Détails de la requête :
1 : Récupération des minimums et maximums de chaque attribut dans le but de les utiliser plus tard pour la normalisation des données
2 : Choix de la chanson que notre utilisateur a aimé
3 : Récupération de toutes les autres chansons de la base de données
4 : Calcul de la similarité
5 : Recherche du nom de l'artiste et de l'album en se basant des relations qui existent entre les chansons et les artistes et les albums

Requêtes re de recommandation avec le cosinus de similarité

1
```
match(TT:Tracks) with min(TT.acousticness) as min_ac, max(TT.acousticness) as max_ac,
min(TT.danceability) as min_dc, max(TT.danceability) as max_dc,
min(TT.energy) as min_en, max(TT.energy) as max_en,
min(TT.instrumentalness) as min_ins, max(TT.instrumentalness) as max_ins,
min(TT.liveness) as min_li, max(TT.liveness) as max_li,
min(TT.loudness) as min_lo, max(TT.loudness) as max_lo,
min(TT.speechiness) as min_sp, max(TT.speechiness) as max_sp,
min(TT.tempo) as min_tp, max(TT.tempo) as max_tp,
min(TT.valence) as min_va, max(TT.valence) as max_va
```

2
```
match(T:Tracks) with T, min_ac, max_ac, min_dc, max_dc, min_en, max_en, min_ins,
max_ins, min_li, max_li, min_lo, max_lo, min_sp, max_sp, min_tp, max_tp, min_va,
max_va where T.id = "3aEEfw8PNGpqdWFKOMPeJN"
```

3
```
match(C:Tracks) with T, C, min_ac, max_ac, min_dc, max_dc, min_en, max_en, min_ins,
max_ins, min_li, max_li, min_lo, max_lo, min_sp, max_sp, min_tp, max_tp, min_va,
max_va where C.id <> "3aEEfw8PNGpqdWFKOMPeJN"
with T, C, COUNT(C) AS totalMusic,
```

4
```
SUM(
((T.acousticness-min_ac)/(max_ac-min_ac))*((C.acousticness-min_ac)/(max_ac-min_ac))+
((T.danceability-min_dc)/(max_dc-min_dc))*((C.danceability-min_dc)/(max_dc-min_dc))+
((T.energy-min_en)/(max_en-min_en))*((C.energy-min_en)/(max_en-min_en))+
((T.instrumentalness-min_ins)/(max_ins-min_ins))*((C.instrumentalness-min_ins)/(max_ins-
min_ins))+
((T.liveness-min_li)/(max_li-min_li))*((C.liveness-min_li)/(max_li-min_li))+
((T.loudness-min_lo)/(max_lo-min_lo))*((C.loudness-min_lo)/(max_lo-min_lo))+
((T.speechiness-min_sp)/(max_sp-min_sp))*((C.speechiness-min_sp)/(max_sp-min_sp))+
((T.tempo-min_tp)/(max_tp-min_tp))*((C.tempo-min_tp)/(max_tp-min_tp))+
((T.valence-min_va)/(max_va-min_va))*((C.valence-min_va)/(max_va-min_va))) as
total_prod_ct,
```

```
SQRT(
((T.acousticness-min_ac)/(max_ac-min_ac))^2+
((T.danceability-min_dc)/(max_dc-min_dc))^2+
 ((T.energy-min_en)/(max_en-min_en))^2+
((T.instrumentalness-min_ins)/(max_ins-min_ins))^2+
 ((T.liveness-min_li)/(max_li-min_li))^2+
((T.loudness-min_lo)/(max_lo-min_lo))^2+
((T.speechiness-min_sp)/(max_sp-min_sp))^2+
((T.tempo-min_tp)/(max_tp-min_tp))^2+
((T.valence-min_va)/(max_va-min_va))^2) as a_total,
```

5

```
SQRT(
((C.acousticness-min_ac)/(max_ac-min_ac))^2+
((C.danceability-min_dc)/(max_dc-min_dc))^2+
((C.energy-min_en)/(max_en-min_en))^2+
((C.instrumentalness-min_ins)/(max_ins-min_ins))^2+
 ((C.liveness-min_li)/(max_li-min_li))^2+
((C.loudness-min_lo)/(max_lo-min_lo))^2+
 ((C.speechiness-min_sp)/(max_sp-min_sp))^2+
((C.tempo-min_tp)/(max_tp-min_tp))^2+
((C.valence-min_va)/(max_va-min_va))^2) as b_total
```

6

```
with T,C, total_prod_ct/(a_total * b_total) as sim order by sim desc Limit 3
```

7

```
match (m:Tracks {id: C.id})<-[]-(A:Artists)
match (Trk:Tracks {id: C.id})<-[]-(Al:Albums)

return Al.name, C.name, COLLECT (A.name) as Artist_name, C.country, C.type, sim;
```

8

Résultat de la requête

Al.name	C.name	Artist_name	C.country	C.type	sim
"80-luvun suursuosikit"	"Tahdon sinut"	["Meiju Suvas"]	"FI"	"track"	0.9959843913842725
"Señor Django"	"Señor Django"	["Kpoint"]	"BE"	"track"	0.995246857093841
"Moonshine Freeze"	"Moonshine Freeze"	["This Is The Kit"]	"FI"	"track"	0.9928639911682611

Détails de la requête :
1 : Récupération des minimums et maximums de chaque attribut dans le but de les utiliser plus tard pour la normalisation des données
2 : Choix de la chanson que notre utilisateur a aimé
3 : Récupération de toutes les autres chansons de la base de données
4 : Calcul du numérateur de la formule de cosinus de similarité sur l'ensemble des propriétés
5 & 6 : Calcul du dénominateur de la formule de cosinus de similarité sur l'ensemble des propriétés
7 : Calcul de la similarité

8 : Recherche du nom de l'artiste et de l'album en se basant des relations qui existent entre les chansons et les artistes et les albums