INF8810 - Traitement et analyse de données massives

Projet de Session 2

Adil, Labiad:	LABA06058403
Amadou II, DIALLO:	DIAA05328802
Da silva Gomes, Daniel Alexandre:	DASD06308309
ISSA BABAN CHAWAI, ABDOULAYE:	ISSA23038902

8 juillet 2023

Table des matières

1 La source de données **1**

2 Prétraitement des données **2**

3 Chargement des données **3**
3.1 Annonces (*Listings*) 3
3.1.1 Création des indexes 3
3.1.2 Création des contraintes 4
3.1.3 Création des nœuds Annonce 4
3.1.4 Création des relations entre Annonce et Quartier . . . 5
3.1.5 Création des relations entre Annonce et Aménités . . . 5
3.1.6 Création des nœuds Hôte 6
3.1.7 Création des relations entre Annonce et Hôte 6
3.2 Avis (*Reviews*) 7
3.2.1 Création des indexes 7
3.2.2 Création des contraintes 7
3.2.3 Création des nœuds Usager 7
3.2.4 Création des nœuds Avis 7
3.2.5 Création des relations entre Usager, Avis et Annonce . 8
3.3 Représentation graphique du graphe 8

4 Système de recommandation **9**
4.1 Analyse 9
4.1.1 Les quartiers les plus populaires 9
4.1.2 Unicité de l'utilisateur 10
4.1.3 Les quartiers fréquentés par l'utilisateur 11
4.1.4 Les aménités préférées de l'utilisateur 11
4.1.5 Les utilisateurs ayant des avis dans les quartiers communs 12
4.1.6 Les aménités en commun entre l'utilisateur et les annonces partagées 13
4.2 Recommandation 14

1 La source de données

Le dataset *Airbnb* utilisé dans ce projet a été téléchargé à partir du site http://insideairbnb.com/get-the-data. Les fichiers téléchargés sont **listings** et **reviews**. Ces fichiers contiennent des informations sur les annonces de logements et les avis associés sur la plateforme Airbnb.

Le fichier **listings.csv** contient des détails sur les annonces de logements disponibles sur Airbnb. Chaque ligne du fichier représente une annonce spécifique et comprend des informations telles que l'**ID** de l'annonce, le **titre**, la **description**, le **type de logement**, le **quartier**, le **prix**, les **équipements disponibles**, les **informations sur l'hôte**, etc. Ce fichier est utile pour analyser les caractéristiques des logements proposés sur *Airbnb* et comprendre les tendances du marché.

Le fichier **reviews.csv** contient les avis laissés par les utilisateurs sur les logements qu'ils ont réservés via *Airbnb*. Chaque ligne du fichier représente un avis spécifique et comprend des informations telles que l'**ID** de l'avis, l'**ID** de l'utilisateur, l'**ID** de l'annonce, la **date** de l'avis, le **texte** de l'avis, etc. Ce fichier permet d'analyser les opinions et les expériences des utilisateurs concernant les logements *Airbnb* et peut fournir des informations sur la satisfaction des clients, la qualité des logements, etc.

Ces deux fichiers peuvent être utilisés conjointement pour effectuer des analyses approfondies sur le marché *Airbnb*, telles que l'étude des préférences des utilisateurs, l'identification des quartiers les plus populaires, l'examen des tendances de tarification, l'analyse des sentiments des avis, etc.

Il est important de noter que les données contenues dans ces fichiers sont générées par des utilisateurs d'*Airbnb* et peuvent être soumises à certaines limitations ou biais. Il est donc nécessaire d'effectuer une exploration et une validation des données avant d'en tirer des conclusions ou des recommandations.

En résumé, le dataset *Airbnb* comprend les fichiers csv **listings** et **reviews**. Ces fichiers fournissent des informations sur les annonces de logements et les avis associés, ce qui permet d'effectuer des analyses sur le marché *Airbnb* et les expériences des utilisateurs.

2 Prétraitement des données

Avant de commencer l'analyse des données, nous avons effectué plusieurs transformations avec le language **Python** pour extraire des informations pertinentes et améliorer la qualité des données. Ces transformations nous ont permis de mieux comprendre les caractéristiques des annonces de logements et d'optimiser les résultats de notre analyse. Pour plus de détails sur l'exécution du code, veuillez consulter la documentation disponible dans le fichier **README**.

L'une des premières observations que nous avons faites lors de l'analyse du dataset *Airbnb* était que de nombreux enregistrements avaient des valeurs manquantes pour les attributs **beds**, **bathrooms** et **bedrooms**. Cependant, nous avons remarqué que ces informations étaient disponibles dans la colonne **name** de chaque annonce de logement.

Pour remédier à cette situation, nous avons effectué une transformation en utilisant des expressions régulières pour extraire les informations sur les lits, les chambres et les salles de bains à partir de la colonne **name** et combler les valeurs manquantes. Cette approche nous a permis d'obtenir des données structurées et cohérentes sur ces attributs essentiels des logements.

Nous avons également effectué des transformations sur les colonnes **amenities** et **host_verifications**. Ces colonnes contenaient des données sous forme de listes. Pour faciliter l'analyse, nous avons transformé ces listes en chaînes de caractères séparées par des "|" (pipe). Cette transformation nous a permis de mieux comprendre les équipements proposés par les logements et les vérifications effectuées par les hôtes.

D'autre part, nous avons constaté la présence d'enregistrements provenant d'autres villes et même d'autres pays. Afin de restreindre notre analyse à des données spécifiques à la ville de Montréal, nous avons procédé à un filtrage basé sur l'emplacement en se basant sur la colonne **host_location**. Cette étape nous a permis de sélectionner uniquement les données liées à la ville de Montréal.

En résumé, les transformations appliquées sur le dataset ont permis d'extraire des informations clés sur les lits, les chambres et les salles de bains, de remplir les valeurs manquantes, de structurer les données sur les équipements et les vérifications, ainsi que de filtrer les données par emplacement.

Ces transformations nous ont fourni un ensemble de données de meilleure qualité, prêt à être utilisé pour notre analyse.

3 Chargement des données

Dans le cadre de notre analyse des données *Airbnb*, nous avons décidé de conserver la plupart des colonnes des jeux de données **listings.csv** et **reviews.csv**. La raison principale de cette décision est que, pour le moment, nous n'avons pas encore défini précisément la requête de recommandation ni le type de recommandation que nous allons utiliser dans notre analyse.

En conservant un large éventail de colonnes, nous nous assurons d'avoir une base de données riche en informations, ce qui nous permettra d'explorer différentes perspectives et approches lors de notre étude. Cela nous donne également la flexibilité nécessaire pour ajuster notre analyse en fonction des découvertes et des besoins qui émergeront au fur et à mesure de l'écriture de notre rapport.

Cependant, il convient de noter que certaines colonnes ont été exclues de notre analyse. Il s'agit des colonnes **description**, **host_about** et **commentaire**. Ces colonnes contiennent principalement du texte avec des caractères spéciaux et ne fournissent pas de valeurs ajoutées à notre analyse.

3.1 Annonces (*Listings*)

3.1.1 Création des indexes

Dans le but d'améliorer la performance des requêtes en accélérant la recherche et la récupération des nœuds et des relations en fonction de certaines propriétés, une approche a été adoptée consistant à créer les indexes avant le chargement des données de *Listings*.

```
CREATE INDEX FOR (h:Host) ON h.host_id;
CREATE INDEX FOR (a:Amenity) ON a.name;
CREATE INDEX FOR (l:Listing) ON l.listing_id;
CREATE INDEX FOR (n:Neighborhood) ON n.neighborhood_id;
```

3.1.2 Création des contraintes

```
CREATE CONSTRAINT FOR (h:Host) REQUIRE h.host_id IS UNIQUE;
CREATE CONSTRAINT FOR (a:Amenity) REQUIRE a.name IS UNIQUE;
CREATE CONSTRAINT FOR (l:Listing) REQUIRE l.lising_id IS UNIQUE;
```

3.1.3 Création des nœuds Annonce

La partie du code présenter plus bas permet de charger les données *listings* en utilisant l'en-tête du fichier et en attribuant l'étiquette **Listing** aux nœuds, ensuite attribuer les propriétés nécessaires. Au total, ce processus crée 5157 nœuds, attribue 5157 étiquettes et définit 118611 propriétés.

```
LOAD CSV WITH HEADERS FROM "file:///listings.csv" AS row FIELDTERMINATOR ','
WITH row WHERE row.id IS  NOT NULL
MERGE (l:Listing {listing_id: row.id})
ON CREATE SET
    l.name                          = row.name,
    l.latitude                      = toFloat(row.latitude),
    l.longitude                     = toFloat(row.longitude),
    l.neighborhood_cleansed         = row.neighborhood_cleansed,
    l.reviews_per_month             = toFloat(row.reviews_per_month),
    l.instant_bookable              = row.instant_bookable,
    l.review_scores_value           = toFloat(row.review_scores_value),
    l.review_scores_location        = toFloat(row.review_scores_location),
    l.review_scores_communication = toFloat(row.review_scores_communication),
    l.review_scores_checkin         = toFloat(row.review_scores_checking),
    l.review_scores_cleanliness     = toFloat(row.review_scores_cleanliness),
    l.review_scores_rating          = toFloat(row.review_scores_rating),
    l.availability_365              = toInteger(row.availability_365),
    l.availability_90               = toInteger(row.availability_90),
    l.availability_60               = toInteger(row.availability_60),
    l.availability_30               = toInteger(row.availability_30),
    l.price                         = toInteger(substring(row.price, 1)),
    l.beds                          = toInteger(row.beds),
    l.bedrooms                      = toInteger(row.bedrooms),
    l.bathrooms                     = toInteger(row.bathrooms),
```

```
    l.accommodates                  = toInteger(row.accommodates),
    l.room_type                     = row.room_type,
    l.property_type                 = row.property_type;
```

3.1.4 Création des relations entre Annonce et Quartier

La partie du code présenter plus bas permet de charger les données *listings* en utilisant l'en-tête du fichier et en attribuant l'étiquette **Neighborhood** aux nœuds, ensuite relier chaque noeud **Listing** à un quartier correspondant et attribuer les propriétés nécessaires. Ce processus crée 32 nœuds, attribue 32 étiquettes, définit 4042 propriétés et crée 5157 relations **IN_NEIGHBORHOOD** entre les noeuds **Listing** et **Neighborhood**.

```
LOAD CSV WITH HEADERS FROM "file:///listings.csv" AS row FIELDTERMINATOR ','
WITH row WHERE row.id IS  NOT NULL
MATCH (l:Listing {listing_id: row.id})
MERGE (n:Neighborhood {
            neighborhood_id: COALESCE(row.neighbourhood_cleansed, "NA")
            })
SET n.name = row.neighbourhood
MERGE (l)-[:IN_NEIGHBORHOOD]->(n);
```

3.1.5 Création des relations entre Annonce et Aménités

La partie du code présenter plus bas permet de charger les données *listings* en utilisant l'en-tête du fichier, créer des nœuds **Amenity** à partir de la colonne *amenities* du fichier, ensuite relier chaque noeud **Listing** à une aménité correspondante et définit les propriétés nécessaires. Ce processus crée 2040 nœuds, attribue 2040 étiquettes, défini 2040 propriétés et crée 180403 relations **HAS** entre les nœuds **Listing** et **Amenity**.

```
LOAD CSV WITH HEADERS FROM "file:///listings.csv" AS row FIELDTERMINATOR ','
WITH row WHERE row.id IS  NOT NULL
MATCH (l:Listing {listing_id: row.id})
UNWIND split(row.amenities,'|') as amenity
MERGE (a:Amenity {name: trim(amenity)})
MERGE (l)-[:HAS]->(a);
```

3.1.6 Création des nœuds Hôte

La partie du code présenter plus bas permet de charger les données *listings* en utilisant l'en-tête du fichier et en attribuant l'étiquette **Host** aux nœuds, ensuite attribuer les propriétés nécessaires. Au total, ce processus crée 2219 nœuds, attribue 2219 étiquettes et définit 26628 propriétés.

```
LOAD CSV WITH HEADERS FROM "file:///listings.csv" AS row FIELDTERMINATOR ','
WITH row WHERE row.host_id IS NOT NULL
MERGE (h:Host {host_id: row.host_id})
ON CREATE SET
    h.name             = row.host_name,
    h.verifications    = row.host_verifications,
    h.listings_count   = toInteger(row.host_listings_count),
    h.acceptance_rate  = toFloat(row.host_acceptance_rate),
    h.host_since       = row.host_since,
    h.url              = row.host_url,
    h.response_rate    = row.host_response_rate,
    h.superhost        = row.host_is_superhost,
    h.location         = row.host_location,
    h.verified         = row.host_identity_verified,
    h.image            = row.host_picture_url
```

3.1.7 Création des relations entre Annonce et Hôte

La partie du code présenter plus bas permet de charger les données *listings* en utilisant l'en-tête du fichier, créer des relations **HOSTED_BY** entre les nœuds **Host** et **Listing** en utilisant l'attribut *host_id*. Ce processus crée 5157 relations.

```
LOAD CSV WITH HEADERS FROM "file:///listings.csv" AS row FIELDTERMINATOR ','
WITH row WHERE row.listing_id IS NOT NULL
MATCH (h:Host {host_id: row.host_id})
```

```
MATCH (l:Listing {listing_id: row.id})
MERGE (h)-[:HOSTED_BY]->(l);
```

3.2 Avis (*Reviews*)

3.2.1 Création des indexes

```
CREATE INDEX FOR (u:User) ON u.user_id;
CREATE INDEX FOR (r:Review) ON r.review_id;
```

3.2.2 Création des contraintes

```
CREATE CONSTRAINT FOR (u:User) REQUIRE u.user_id IS UNIQUE;
CREATE CONSTRAINT FOR (r:Review) REQUIRE r.review_id IS UNIQUE;
```

3.2.3 Création des nœuds Usager

La partie du code présenter plus bas permet de charger les données *reviews* en utilisant l'en-tête du fichier, créer des nœuds **User** correspondants en l'attribuant les propriétés *reviewer_id* et *reviewer_name*. Ce processus créé 237470 nœuds, attribue 237470 étiquettes et défini 504033 propriétés.

```
LOAD CSV WITH HEADERS FROM "file:///reviews.csv" AS row FIELDTERMINATOR ','
WITH row WHERE row.listing_id IS NOT NULL
MERGE (u:User {user_id: row.reviewer_id})
SET u.name = row.reviewer_name;
```

3.2.4 Création des nœuds Avis

La partie du code présenter plus bas permet de charger les données *reviews* en utilisant l'en-tête du fichier, créer des nœuds **Review** correspondants en l'attribuant les propriétés *review_id* et *date*. Ce processus créé 266563 nœuds, attribue 266563 étiquettes et défini 533126 propriétés.

```
LOAD CSV WITH HEADERS FROM "file:///reviews.csv" AS row FIELDTERMINATOR ','
WITH row WHERE row.listing_id IS NOT NULL
MERGE (r:Review {review_id: row.id})
SET r.date = row.date;
```

3.2.5 Création des relations entre Usager, Avis et Annonce

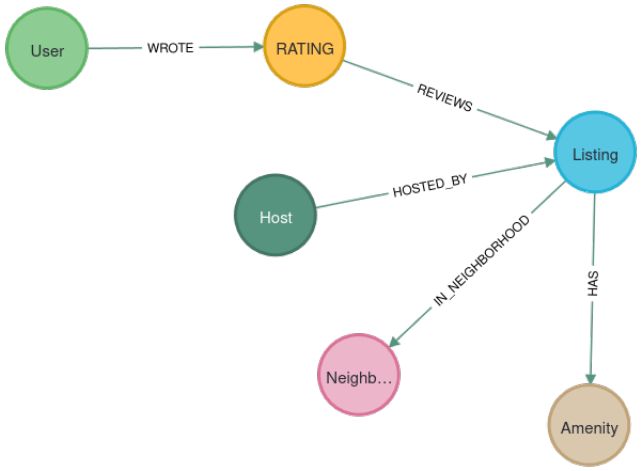
La partie du code présenter plus bas permet de charger les données *reviews* en utilisant l'en-tête du fichier, créer des relations entre les nœuds **User**, **Review** et **Listing** correspondants. Ce processus crée 397230 relations.

```
LOAD CSV WITH HEADERS FROM "file:///reviews.csv" AS row FIELDTERMINATOR ','
WITH row WHERE row.listing_id IS NOT NULL
MATCH (u:User {user_id: row.reviewer_id})
WITH u, row
MATCH (r:Review {review_id: row.id})
WITH u, r, row
MATCH (l:Listing {listing_id: row.listing_id})
MERGE (u)-[:WROTE]->(r)
MERGE (r)-[:REVIEWS]->(l);
```

3.3 Représentation graphique du graphe

Après avoir créé les nœuds et les relations, nous pouvons voir le graphique général de la base de données avec la requête suivant :

```
CALL db.schema.visualization
```



Note 1 *Nous avons rencontré un problème lors de l'affichage du label* ***Review*** *sur les nœuds dans notre graphe. Malheureusement, malgré nos efforts, le label* ***Review*** *ne s'affichait pas correctement. Pour remédier à cette situation, nous avons changer le nom du label en* ***Rating****. Cela nous a permis de visualiser correctement le graphique général de la base de données. Veuillez noter que le code* Cypher *utilisé pour les requêtes continue à faire référence au label* ***Review****. Nous avons donc simplement ajusté l'affichage pour des raisons de visualisation.*

4 Système de recommandation

4.1 Analyse

4.1.1 Les quartiers les plus populaires

La requête ci-dessous permet de vérifier les quartiers les plus populaires sur la base des notes moyennes totales des avis.

```
MATCH (l:Listing)-[:IN_NEIGHBORHOOD]->(n:Neighborhood)
WITH n.neighborhood_id AS neighborhood, COALESCE(AVG(l.review_scores_rating), 0) AS average_rating
WHERE average_rating > 0
RETURN neighborhood, average_rating
ORDER BY average_rating DESC
LIMIT 10
```

neighborhood	average_rating
"Baie-d'Urfé"	5.0
"Dorval"	4.91
"Sainte-Anne-de-Bellevue"	4.890000000000001
"Beaconsfield"	4.8740000000000006
"L'Île-Bizard-Sainte-Geneviève"	4.860000000000001
"Westmount"	4.849333333333335
"Saint-Léonard"	4.815833333333334
"Montréal-Ouest"	4.811428571428572
"Rosemont-La Petite-Patrie"	4.795655976676386
"LaSalle"	4.79156862745098

4.1.2 Unicité de l'utilisateur

La requête ci-dessous permet vérifier la quantité de données disponibles pour l'utilisateur **97763164**.

```
MATCH (u:User {user_id: "97763164"})
RETURN count(*) AS user_data_count
```

user_data_count
1

4.1.3 Les quartiers fréquentés par l'utilisateur

La requête ci-dessous permet de compter le nombre de fois que l'utilisateur **97763164** a visité chaque quartier. Cela nous donnera une idée des quartiers préférés de l'utilisateur.

```
MATCH (u:User {user_id: "97763164"})-[:WROTE]->(:Review)-[:REVIEWS]->(:Listing)-[:IN_NEIGHBORHOOD]->(n:Neighborhood)
RETURN n.neighborhood_id, COUNT(*) AS visit_count
ORDER BY visit_count DESC
```

n.neighborhood_id	visit_count
"Côte-des-Neiges-Notre-Dame-de-Grâce"	19

4.1.4 Les aménités préférées de l'utilisateur

La requête ci-dessous permet de compter le nombre de fois que l'utilisateur **97763164** a mentionné chaque aménité dans ses avis. Cela nous permettra de comprendre les aménités préférées de l'utilisateur.

```
MATCH (u:User {user_id: "97763164"})-[:WROTE]->(:Review)-[:REVIEWS]->(l:Listing)-[:HAS]->(a:Amenity)
RETURN a.name, COUNT(*) AS amenity_count
ORDER BY amenity_count DESC
LIMIT 10
```

a.name	amenity_count
"Washer"	19
"Dryer"	19
"Heating"	19
"Lock on bedroom door"	19
"Essentials"	19
"Kitchen"	19
"Hangers"	19
"Smoke alarm"	19
"Wifi"	18
"Shampoo"	14

4.1.5 Les utilisateurs ayant des avis dans les quartiers communs

La requête ci-dessous permet de récupère les utilisateurs qui ont laissé un avis dans les mêmes quartiers que l'utilisateur **97763164**.

```
MATCH (u:User {user_id: "97763164"})-[:WROTE]->(:Review)-[:REVIEWS]->(:Listing)-[:IN_NEIGHBORHOOD]->(n:Neighborhood)
WITH u, COLLECT(DISTINCT n) AS neighborhoods
MATCH (u2:User)-[:WROTE]->(:Review)-[:REVIEWS]->(:Listing)-[:IN_NEIGHBORHOOD]->(n2:Neighborhood)
WHERE n2 IN neighborhoods AND u2 <> u
RETURN DISTINCT u2.user_id
LIMIT 10
```

u2.user_id
"165621804"
"429727271"
"117561454"
"253054615"
"4802738"
"214022447"
"161925758"
"118643191"
"183400133"
"252738397"

4.1.6 Les aménités en commun entre l'utilisateur et les annonces partagées

La requête ci-dessous permet d'identifier les aménités en commun entre l'utilisateur **97763164** et les annonces partagées.

```
MATCH (u:User {user_id: "97763164"})-[:WROTE]->(:Review)-[:REVIEWS]->(l:Listing)-[:IN_NEIGHBORHOOD]->(:Neighborhood)
WITH u, COLLECT(DISTINCT l) AS user_listings
MATCH (u)-[:WROTE]->(:Review)-[:REVIEWS]->(l2:Listing)-[:IN_NEIGHBORHOOD]->(:Neighborhood)
WHERE l2 IN user_listings
MATCH (l2)-[:HAS]->(a:Amenity)
RETURN DISTINCT a.name AS amenity
LIMIT 10
```

amenity
"Wifi"
"Heating"
"Essentials"
"Hair dryer"
"Lock on bedroom door"
"Host greets you"
"Shampoo"
"Hangers"
"Smoke alarm"
"Washer"

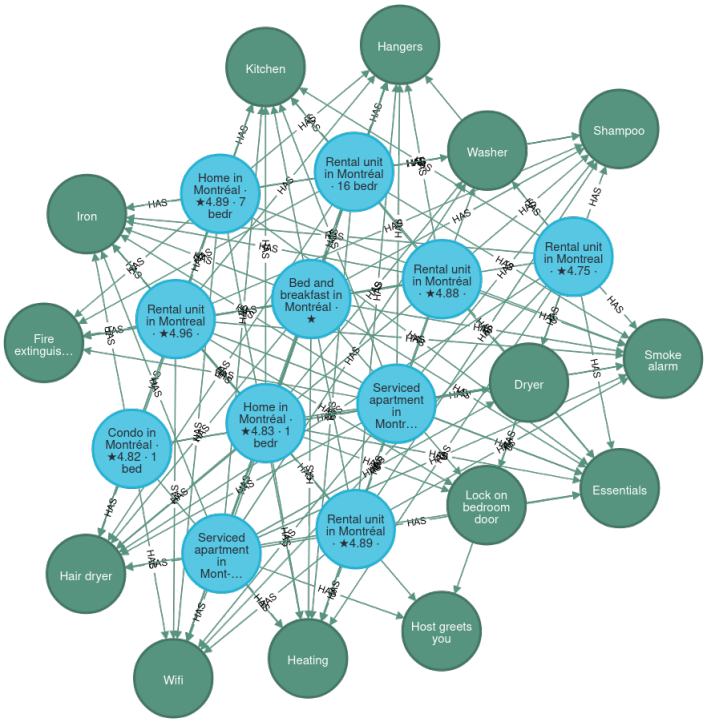
4.2 Recommandation

La requête ci-dessous recommande des annonces similaires à un utilisateur spécifié en se basant sur les aménités partagées. Les annonces sont sélectionnés dans les mêmes quartiers que ceux associés aux annonces sur lesquels l'utilisateur a donné un avis. Les annonces sont ensuite triés en fonction du nombre d'aménités en commun avec l'utilisateur, et les 5 meilleurs résultats sont renvoyés. Cette approche de recommandation vise à suggérer des annonces les plus pertinentes qui ont des caractéristiques similaires à ceux que l'utilisateur a appréciés.

```
MATCH (u:User {user_id: "97763164"})-[:WROTE]->(:Review)-[:REVIEWS]->(:Listing)-[:IN_NEIGHBORHOOD]->(n:Neighborhood)
WITH u, COLLECT(DISTINCT n) AS neighborhoods
MATCH (u)-[:WROTE]->(:Review)-[:REVIEWS]->(l:Listing)-[:IN_NEIGHBORHOOD]->(n:Neighborhood)
WHERE n IN neighborhoods
MATCH (l:Listing)-[:HAS]->(a:Amenity)
WITH l, COLLECT(DISTINCT a) AS userAmenities
```

```
MATCH (rec:Listing)-[:HAS]->(recAmenity:Amenity)
WHERE recAmenity IN userAmenities AND rec <> l
RETURN rec.listing_id, COUNT(DISTINCT recAmenity) AS score
ORDER BY score DESC
LIMIT 10
```

rec.listing_id	score
"36510908"	33
"50662164"	32
"24697639"	31
"34720098"	31
"51921841"	31
"820093019174634943"	31
"11712825"	31
"11860692"	31
"744459"	30
"29732225"	30



Cette partie de la requête récupère le nœud **User** correspondant à l'**ID** spécifié et trouve les quartiers (noeuds **Neighborhood**) associés aux annonces (noeuds **Listing**) sur lesquels l'utilisateur a écrit des critiques.

```
MATCH (u:User {user_id: "97763164"})-[:WROTE]->\
    (:Review)-[:REVIEWS]->(:Listing)-[:IN_NEIGHBORHOOD]->(n:Neighborhood)
```

La clause **WITH** collecte les quartiers **Neighborhood** dans une liste distincte appelée *neighborhoods* et les transmet à l'étape suivante de la requête.

```
WITH u, COLLECT(DISTINCT n) AS neighborhoods
```

Cette partie de la requête récupère les annonces (nœuds **Listing**) associés aux quartiers de la liste *neighborhoods* pour tous les utilisateurs qui ont laissé des avis sur ces annonces.

```
MATCH (u)-[:WROTE]->(:Review)-[:REVIEWS]->\
    (l:Listing)-[:IN_NEIGHBORHOOD]->(n:Neighborhood)
```

La clause **WHERE** filtre les annonces en ne conservant que ceux qui se trouvent dans les quartiers de la liste *neighborhoods*.

```
WHERE n IN neighborhoods
```

Cette partie de la requête récupère les aménités (nœuds **Amenity**) associées aux annonces sélectionnés précédemment.

```
MATCH (l:Listing)-[:HAS]->(a:Amenity)
```

La clause **WITH** collecte les aménités dans une liste distincte appelée *userAmenities* et les transmet à l'étape suivante de la requête.

```
WITH l, COLLECT(DISTINCT a) AS userAmenities
```

Cette partie de la requête récupère d'autres annonces (nœuds **Listing**) qui ont des aménités en commun avec celles de l'utilisateur.

```
MATCH (rec:Listing)-[:HAS]->(recAmenity:Amenity)
```

La clause **WHERE** sélectionne les annonces en filtrant uniquement celles qui partagent des aménités avec celles de l'utilisateur, mais qui sont distinctes des annonces (noeuds **Listing**) pour lesquelles l'utilisateur a laissé un avis. Cette condition permet de recommander des annonces similaires à l'utilisateur, tout en excluant celles qu'il a déjà évaluées.

```
WHERE recAmenity IN userAmenities AND rec <> l
```

Cette partie de la requête renvoie l'**ID** de l'annonce recommandée et compte le nombre d'aménités en commun avec l'utilisateur, appelé *score*.

```
RETURN rec.listing_id, COUNT(DISTINCT recAmenity) AS score
```

La clause **ORDER BY** trie les résultats par ordre décroissant du *score*, ce qui signifie que les annonces avec le plus grand nombre d'aménités en commun seront les premiers.

```
ORDER BY score DESC
```

La clause **LIMIT** limite les résultats à 5 annonces recommandés.

```
LIMIT 5
```