Bases de Données Large Echelle - DM1

Prosio Giulia

October 16, 2022

Abstract

1 Introduction

Le but de ce rapport est de détailler ma résolution des problèmes et l'analyse des données proposée lors des travaux du projet, tout en expliquant la raison de mes choix de code et la théorie derrière les fonctions utilisées.
Le premier projet était axé sur l'analyse générale des données et l'utilisation de SQL dans un contexte de grande quantité de données.
Avec le second projet, nous avons déplacé l'accent sur les modèles multidimensionnels et les requêtes associées. Ici les types d'analyse proposés ont été développés dans deux directions : l'agrégation de données et les fenêtres.

2 TP1

Ce TP se concentre sur la préparation de données et des différentes techniques d'analyse de données avec le langage sql, supportées par UDF en python.

2.1 Exercice 1

2.1.1 Point 1 - Les 10 POI les plus photographiés

Pour obtenir le résultat du exercice 1 j'ai fusionné les colonnes de deux tables différentes, user_visits et POI, avec l'expression WHERE a = b, étant a et b le poiID dans les deux tableaux.
J'ai donc utilisé l'expression ORDER BY poiFreq desc et LIMIT 10 pour retrouver les 10 POI les plus photographiés.

2.1.2 Point 2 - Les visites avec date détaillée

Pour definir la table contenant les visites avec la date détaillée j'ai utilisé la fonction from_unixtime(), avec en paramètre la date (au format unix) et la partie de la date qui m'intéressait (année, mois, jour, heure, minute et seconde).

2.1.3 Point 3a - Le nombre de POI par utilisateur

Pour obtenir la tuple (userID, nombre de POIs) trié par nombre décroissant de POI, j'ai utilisé la fonction count(distinct poiID), pour éviter de compter les doublons.

2.1.4 Point 3b - Le nombre de POI par séquence

Pour définir la table Seq3plus(seqID, nbPoi) correspondant aux séquences qui contiennent au moins 3 POI distincts, j'ai utilisé la fonction COUNT(DISTINCT ..) pour m'assurer de ne pas compter les doublons.
J'ai donc utilisé ORDER BY pour présenter le résultat par ordre décroissant des POIs visités.

2.1.5 Point 4 - Trajectoire

La résolution de cet exercice se compose de deux parties : j'ai d'abord défini une fonction en python, puis j'ai utilisé cette fonction dans la requête SQL.
J'ai ainsi utilisé une UDF, une fonction définie par l'utilisateur qui permet d'effectuer des opérations spécifiques sur les données.
Cette UDF reçoit en particulier comme paramètre d'entrée une liste de POI avec leurs dates de visite respectives et produit une liste de POI, triée chronologiquement par rapport à la date de visite.
J'utilise ensuite la commande "spark.udf.register()" pour définir le nom de la fonction, la fonction associée et le type de sortie.
Dans la requête sql, afin d'utiliser mon UDF, j'ai dû transformer les colonnes poiID et date en une liste contenant les deux valeurs. Pour ce faire, j'ai utilisé deux fonctions : ARRAY_AGG() qui renvoie le array correspondant à la colonne de la table, et ARRAYS_ZIP() qui renvoie un array unique pour les deux arrays créés (date et poiID).

2.1.6 Point 5 - Transitions

Dans cet exercice, j'utilise un UDF pour définir, à partir d'un tableau de POIs, les transitions d'un POI à un autre. De cette façon, je suis en mesure de montrer le tuple (seqID, poi1, poi2) dans la requête SQL, où la réponse est le seqID, le poi1 de départ et le poi2 d'arrivée.

2.1.7 Point Distance entre les POIs

La demande pour cet exercice était de créer un tableau contentant deux POIs et leur distance.
Le tableau POI contient les POIs et leur coordonnées géographiques (latitude et longitude).
Pour calculer la distance entre deux POIs, j'ai donc utilisé la formule de Haversine, en profitant des fonctions fournies par SQL asin(), sqrt(), power(), sin() et cos().

2.1.8 Point 6 - DuréeVisitePOI

On doit calculer la durée moyenne de visite d'un POI.
Pour faire ça j'ai utilisée les formules min(date), max(date) et avg().
De cette façon, j'ai pris la date minimale et maximale à laquelle l'utilisateur était à un endroit de la séquence et j'ai pu calculer le temps qu'il a passé à ce POI.

2.2 Exercice 2

2.2.1 Point 1 - Geonames

Cet exercice comporte deux points : a et b. La première étape nécessite de définir la table Geonames2 en précisant le schéma nom et type des attributs.
J'ai donc utilisé la formule CAST(.. as type) pour compléter la demande.
Le deuxième point est d'analyser le tableau uniquement pour les POIs situés canada.
La table Geonames contient une colonne country_code, donc avec la fonction where country_code = 'CA' je peux limiter la table.

2.2.2 Point 2 - Association entre les POI et Geonames

Dans cet exercice, il est demandé d'associer les POI étudiés aux géonames. Pour ce faire, j'ai considéré les coordonnées géographiques (latitude et longitude) des deux.

En particulier, j'ai créé une association si la différence entre les latitudes et la différence entre les longitudes était inférieure ou égale à un degré

3 TP2-3

Ce TP se concentre sur l'utilisation des fenêtres et des fonctions associées pour le dépannage et l'analyse des données contenues dans les tableaux proposés.

3.1 Exercice 1

3.1.1 Point 1 - Identifier les thèmes

Les différents points d'intérêt du tableau peuvent appartenir à l'un des six thèmes possibles.
La requête de l'exercice était d'extraire les noms des différents thèmes du tableau et d'associer à chacun d'eux un nombre entier dans l'ordre croissant.

Pour ce faire, j'ai utilisé la fonction row_number(), qui nécessite le paramètre OVER(). OVER() est une fonction de fenêtrage utilisée pour considérer une fenêtre de tuples dans la table - dans ce cas, la colonne Themes dans user_visits.

3.1.2 Point 2 - Classement des séquences par leur plus grand nombre de POI distincts

La requête de cet exercice était de montrer avec un classement la séquence de visites d'un utilisateur contenant le plus grand nombre de POIs distincts.
Dans ce cas, j'ai utilisé la fonction rank() OVER().
En ce qui concerne la fonction OVER(), le paramètre dans ce cas était le tuple (seqID, nombre de POI distincts différents visités dans cette séquence).

3.1.3 Point 2 - Identifier les check-ins

Cet exercice se compose de trois sous-questions successives.
Le premier point nécessite de définir une table Visit1 telle qu'il n'y ait pas de doublons dans le triplet (seqID, thenID, dates). Tout ce dont j'avais besoin pour faire cela était une commande de sélection distincte.
La deuxième étape consistait à prouver qu'il existait des séquences dans lesquelles plusieurs POI avaient été visités le même jour. Pour ce faire, j'ai effectué un COUNT() sur les résultats du tableau précédent.
Le point final du problème était alors de définir la table Visits2 contenant une colonne 'num' le numéro d'ordre d'un POI dans la séquence.
Pour cela j'ai utilisé la fonction rank() OVER() la partition de chaque seqID triée par date

3.1.4 Point 3 - Identifier les Visites de POI

Cet exercice comporte trois points : a, b, c.
La première partie de l'exercice (3a) nécessite de créer la table Visite3a où chaque POI d'une séquence est associé au POI visité avant - colonne nommée "precedent".
Pour ce faire, j'ai utilisé la fonction rank() OVER() une partition des seqID considérés, ordonnés par date pour obtenir la séquence d'entiers représentant l'ordre des POIs visités dans une séquence. Ensuite, j'ai utilisé la fonction lag() prenant comme paramètre le poiID précédant et le poiID courant avec une distance 1.
La fonction lag() est suivi par OVER(), ayant les mêmes paramètres que la fonction définie ci-dessus.
Ensuite, j'ai procédé à la création d'un nouveau tableau, qui affichait également le précédent point d'intérêt visité dans une séquence, mais avec également une nouvelle colonne avec les valeurs 0 ou 1.
Si un tuple de la table représente le début d'une série de photos prises dans le même POI, alors sa valeur va être mise à 1, sinon elle est mise à 0.
Pour ce faire, j'ai implémenté un CASE WHEN ... THEN ... ELSE ... END, avec comme condition la répétition du POI visité dans une séquence, vérifiée avec la fonction lag(poiID, -1).
Enfin, j'ai créé la table Visite3, ayant comme colonne ajoutée "poiPosition", telle que la poiPosition

vaut i pour le ième POI visité dans une séquence.
Pour créer cette colonne j'ai utilisé la combinaison des deux fonctions: sum et lag avec le CASE WHEN... ALORS... ELSE... END expression.
De cette façon, on obtient une séquence ordonnée par date, avec les POIs visités qui considère si le même POI a été visité plusieurs fois dans une période de temps subséquente.

3.1.5 Point 4 - Durée de visite d'un POI

Dans cet exercice, la table Visite4 est définie comme contenant une nouvelle colonne, appellée "duree", qui calcule la différence entre la première et la dernière photo prise dans un POI d'une séquence et renvoie le temps que le visiteur a passé dans ce POI.
Pour créer cette colonne j'ai utilisé les fonctions min(date) et max(date) avec group by seqID, poiID et poiPosition.

3.1.6 Point 4a - Nombre moyen de visites et nombre moyen de POI dans une séquence

Cet exercice demande le nombre moyen the visites et the POI dans une séquence.
On a vu avec les tables précédents que le nombre de visites peux etre plus grande que le nombre de POI parce que dans une séquence le meme POI peut etre visité plusierus fois.
Donc pour computer les deux j'ai utilisé la fonction avg() prenant comme paramètre la fonction count(distinct poiID) pour le nombre moyen de POI visites et avg() avec comme paramètre count(poiID) pour les visites.

3.1.7 Point 4b - Nombre de séquences selon leur nombre de visites

L'exercice 4b est composé de trois sous-items nécessitant l'utilisation des fenêtre glissantes. Les deux premiers points nécessitent la présentation de la relation entre le nombre de séquences et le nombre de visites ou des POIs distinctes visités.

J'analyse maintenant le tableau mettant en relation le nombre de POI distincts visités et le nombre de séquences qui leur correspondent.
Le tableau se compose de trois colonnes: le nombre de POI distincts,le nombre de séquences dans lesquelles le nombre de POI a été visité et la somme cumulative (c'est-à-dire le nombre de séquences qui ont visité au moins le nombre de POIs proposé).
Pour ce faire, j'ai d'abord créé un tableau intermédiaire faisant correspondre le nombre de séquences dans lesquelles n POIs ont été visités et le nombre n de POIs distincts.
Ensuite, j'ai utilisé ce tableau pour créer une fenêtre glissante et faire la somme cumulative des éléments avec la fonction SUM()OVER(... ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING).

La structure du tableau dans lequel le nombre de POI distincts est remplacé par le nombre de visites est similaire, avec les modifications appropriées liées au fait que le paramètre considéré est different.

Le dernier point requis le nombre de séquences ayant au moins un POI visité 2 fois.
Pour ce faire, j'ai considéré le tableau Visites3b précédemment construit.
Dans ce tableau, j'avais créé une colonne 'debut' qui avait la valeur 1 dans le cas où le POI visité dans la séquence était répété plus tard.
Ainsi, en additionnant toutes les valeurs de début dans la même séquence, j'ai pu déterminer si au moins un POI avait été visité deux fois.

3.1.8 Point 5 - Nombre de visites sur une semaine glissante

Une fenêtre glissante permet de définir l'étude des données internes à un intervalle précis, avec le mot BETWEEN qui permet de définir les bornes de la fonction.

Cet exercice comporte deux points : a et b.
Avec le résultat du point a on peut déduire la table b défini sur la semaine glissante. Avec la table a j'ai defini le nombre de visites qu'un utilisateur a fait chaque jour, avec une simple fonction count().

Après ça, j'ai crée la table b qui renvoie le nombre de POIs visités par l'utilisateur sur une semaine glissante - j'ai utilisée la fonction sum() OVER(ordered by ... ROWS BETWEEN CURRENT ROW AND 6 FOLLOWING).

3.1.9 Point 6

Cet exercice consiste à définir le temps pris dans une séquence pour aller d'un POI à un autre.
Pour ce faire, j'ai d'abord créé un tableau de support dans lequel j'ai défini la colonne "courant" comme étant le POI suivant par rapport à celui de la ligne considérée.
J'ai ensuite créé la table Duree_Deplacement qui, au moyen d'une fonction CASE WHEN, détermine si le POI et le suivant (courant) sont identiques ; si ce n'est pas le cas, la différence de temps entre les deux POI est calculée.
Pour déterminer la colonne poiPosition je me suis référé aux fonctions déjà utilisées dans TP1, ARRAY() pour passer de la colonne à la liste et ARRAY_ZIP() pour fusionner plusieurs tableaux.

3.2 Exercice 2

3.2.1 Point 1

La première étape pour résoudre ce problème est, à mon avis, de considérer la proximité géographique de certains points d'intérêt - et de les considérer comme un seul lieu s'ils ne sont pas très éloignés les uns des autres.

J'ai donc créé une nouvelle vue contenant la fonction DENSE_RANKING() qui associe le même numéro à des photos prises au même endroit selon cette définition.
J'ai ensuite considéré l'intersection des trois exigences restantes : une séquence doit contenir au moins trois points distincts, chacun des points doit être associé à au moins trois utilisateurs et une séquence peut couvrir un jour au maximum.
Pour considérer les 3 points comme distincts, il suffit que les points aient un numéro de dense_ranking différent qui leur soit associé.
Pour le cadre temporel, nous devons limiter l'année, le mois et le jour égaux.
Pour les trois personnes on doit controller qu'il y a au moins 3 identifiants d'utilisateur différents associés.

3.2.2 Point 2

Proposition d'une analyse tridimensionnelle. Je pourrais faire une analyse cubique entre la dimension temporelle, la dimension du nombre d'utilisateurs qui ont visité le POI considéré et la dimension géographique.
En ce qui concerne l'analyse à trois niveaux, je pourrais examiner le plan temporel : année, mois et jour.