Compte rendu du TME1_2 SAM

Anyes TAFOUGHALT

February 25, 2024

1 Introduction

Ce TME implique l'utilisation de la base de données TPC-H benchmark avec les systèmes de gestion de bases de données DuckDB et SQLite. Après la génération des données, nous avons étudié la structure de la base et effectué des requêtes simples. Ensuite, nous avons comparé les performances entre DuckDB et SQLite en exécutant les mêmes requêtes dans les deux systèmes. Nous avons également analysé l'impact de l'optimisation des requêtes dans DuckDB et examiné les plans d'exécution. Enfin, nous avons décidé de focaliser sur la partie concernant les index afin de comprendre leur importance dans l'optimisation des requêtes.

2 Analyse de l'Effet des Index sur les Performances des Requêtes

Dans cette partie du TP, nous avons exploré l'effet des index sur les performances des requêtes dans les bases de données DuckDB et SQLite, en tenant compte des situations où l'utilisation d'un index peut ne pas être nécessaire ou même contre-productive.

2.1 Création d'Index sur o_orderkey

Nous avons commencé par activer l'optimiseur de requêtes avec la commande PRAGMA enable_optimizer dans DuckDB. Ensuite, nous avons créé un index sur la colonne o_orderkey de la table orders avec la commande CREATE INDEX o_orderkey_idx ON orders (o_orderkey). Cela permet d'accélérer les recherches basées sur cette colonne. Pour la requête sélectionnant les valeurs de o_orderkey égales à 500 dans la table orders, nous avons observé que l'index était utilisé avec l'optimiseur activé, réduisant le temps d'exécution à 1 milliseconde. Cependant, en désactivant l'optimiseur, l'index n'était plus utilisé, car la colonne o_orderkey avait seulement 5 valeurs différentes sur un total de 15027, rendant l'utilisation de l'index plus coûteuse que l'analyse séquentielle de la table.

2.2 Requête avec conditon autre qu'un égalité

Nous avons également tester un cas oui une requête cherche à récupérer toutes les lignes où une certaine colonne (o_orderkey) est supérieure à une valeur donnée(1 dans notre test), l'utilisation de l'index peut ne pas être optimale même si cette colonne est indexée de manière unique. L 'optimiseur a décidé qu'une analyse séquentielle des données est plus efficace que l'utilisation de l'index, car elle évite les opérations de recherche coûteuses telles que le pré-chargement des données en mémoire cache (En effet ici on aurait relu sur le disque chaque valeurs séparément car l'index nous renverra beaucoup de rowId).

2.3 Impact des Index sur les Jointures

Après avoir créé un index sur la colonne l_orderkey de la table lineitem, nous avons exécuté une requête impliquant une jointure entre les tables orders et lineitem basée sur l_orderkey. L'index a été utilisé avec succès pour accélérer la jointure, réduisant le temps d'exécution à 7 millisecondes avec l'optimiseur activé. Cependant, en désactivant l'optimiseur, le temps d'exécution a augmenté à 139 millisecondes, démontrant l'importance de l'optimisation des requêtes et de l'utilisation d'index pour améliorer les performances des jointures, même dans des cas où le nombre de valeurs distinctes peut sembler relativement faible par rapport à la taille de la table.

2.4 Sélection avec Index

Dans un dernier scénario, nous avons créé un index sur la colonne c_mktsegment de la table customer, puis exécuté une requête de sélection basée sur cette colonne pour extraire les valeurs de c_custkey lorsque c_mktsegment était égal à 'AUTOMOBILE'. Cependant, malgré la présence de l'index, l'optimiseur de requêtes a choisi de ne pas l'utiliser pour accélérer la recherche. Cette décision découle du fait que la colonne c_mktsegment ne comportait que 5 valeurs distinctes, tandis que la table contenait 15027 enregistrements. Dans ce contexte, l'utilisation de l'index aurait été plus coûteuse en termes de performances, expliquant pourquoi son utilisation n'était pas jugée nécessaire par l'optimiseur.

2.5 Idée et résumé

Dans notre démarche visant à optimiser les performances de la base de données, nous avons entrepris une analyse approfondie des données pour identifier les variables dont les valeurs étaient très mal réparties (En regardant la moyenne des valeurs de chaque attribut notamment ceux dont les valeurs sont uniques afin d'y mettre un index) .

Pour ce faire, nous avons examiné différentes statistiques descriptives telles que la moyenne, les valeurs uniques, ainsi que les quartiles(grace au describe()).

L'idée de rechercher des attributs dont les valeurs sont mal distribuées dans le but d'inciter l'optimiseur à utiliser un index repose sur une compréhension des hypothèses sous-jacentes de l'optimiseur lors de la planification de l'exécution des requêtes. L'optimiseur fait souvent des hypothèses sur la distribution uniforme des données lors de l'estimation des coûts des opérations de recherche. Cela signifie qu'il suppose que les valeurs dans une colonne donnée sont réparties de manière uniforme sur toute la plage des valeurs possibles. Cette hypothèse peut être raisonnable dans de nombreux cas, mais elle peut également conduire à des estimations inexactes des coûts lorsqu'elle est confrontée à des données réellement mal distribuées. Lorsque les valeurs dans une colonne sont fortement concentrées autour de quelques points ou qu'elles présentent une répartition très asymétrique, l'utilisation d'un index peut ne pas être la meilleure stratégie pour accélérer les opérations de recherche. En effet, un index est plus efficace lorsque les valeurs sont réparties de manière uniforme, ce qui permet à la base de données de localiser rapidement les entrées correspondantes. L'idée derrière la recherche d'attributs dont les valeurs sont mal distribuées est donc d'identifier les cas où l'hypothèse d'une distribution uniforme est violée. En créant un index sur une telle colonne, on cherche à inciter l'optimiseur à utiliser cet index,ce même si en réalité il serait plus judicieux de ne pas l'utiliser .

2.6 Conclusion

En conclusion, notre étude met en lumière que l'utilisation d'index n'est pas toujours la meilleure stratégie pour améliorer les performances des requêtes dans les bases de données.Nos analyses ont révélé que l'utilisation d'index peut ne pas être optimale lorsque les requêtes impliquent des opérations autres que des égalités sur des attributs indexés de manière unique, ce qui peut conduire à des décisions de l'optimiseur de requêtes favorisant d'autres méthodes d'accès aux données. Et après plusieurs recherche et documentation nous avons observé que la création d'index sur des colonnes présentant des valeurs mal réparties peut parfois ne pas conduire aux gains de performance escomptés. Plus précisément, dans les cas où les données sont fortement concentrées autour de quelques points ou présentent une distribution asymétrique, l'utilisation d'index peut être moins bénéfique, voire contre-productive, en raison de l'écart par rapport à l'hypothèse d'une distribution uniforme.