M1–Master Informatique – 2024/2025

Bases de données relationnelles

Indexation - Synthèse

1 Index sur la clé primaire

Dans tous les SGBD relationnels, lorsqu'on définit une clé primaire, un index unique (généralement un b-arbre) est automatiquement créé sur cette clé.
A titre d'illustration, le moteur SQL utilise bien l'index B-arbre sur la clé primaire `fac_num` pour exécuter la requête de la question 1.2 (`where clé = valeur`) et une solution alternative à la requête 1.3 (`where clé between valmin and valmax`).
Au TP1, on a mis en évidence l'intérêt d'un index lors d'une sélection (fonction `selection_index`) mais aussi lors d'une jointure (fonctions `jointure_index` et `jointure_double_index`).
Dans le modèle relationnel, on fait souvent des jointures avec des conditions :
`T1.clé_primaire = T2.clé_étrangère`, parfois combinée avec une sélection. C'est pourquoi un index sur la clé primaire est particulièrement utile.
Si on combine plusieurs conditions (question 1.4), on peut bien sûr utiliser un index pour assurer l'une des conditions. On peut aussi utiliser plusieurs index portant sur plusieurs conditions et les combiner en calculant une intersection.
Comme dit dans le sujet de TP2, la problématique de l'utilisation de fonctions posée en question 1.3 est un grand classique en optimisation de requêtes. Si la colonne indexée est "cachée" par une expression ou un appel de fonction, essayer de poser la requête autrement (c'est parfois tout simple, comme par exemple remplacer `(col1+col2)/2 > 10` par `col1 > (20 - col2)` si on dispose d'un index sur `col1`).

2 Création de nouveaux indexes

L'administrateur de la base peut créer d'autres index, ce qui a un impact sur le temps d'exécution des requêtes : il faut estimer le coût de maintenance de ces index, et le gain en performance des requêtes qui bénéficient de ces index.
De manière générale, la création d'un nouvel index est une démarche d'optimisation : elle intervient seulement pour résoudre de mauvais temps d'exécution de telle ou telle requête, et lorsque la reformulation de la requête incriminée n'est pas suffisante.

3 Index multi-colonnes

Il faut parfois choisir entre créer un index sur le couple (col1, col2) ou deux indexes, i.e. un sur chacune des colonnes. Cela dépend du type de requêtes d'interrogation (`select`) et de mises à jour (`insert`, `delete`, `update`) que l'on fait le plus fréquemment.
Un index multi-colonne est un b-arbre sur `(col1, ..., colk)` qui peut être utilisé pour des requêtes où la sélection comporte des conditions d'égalité ou d'intervalle sur `(col1, ..., colk)` ou un préfixe de `(col1, ..., colk)`.

Pour reprendre l'exercice 3 du TP2, on ne peut pas utiliser l'index sur `(lig_facture,lig_produit)` pour une requête dont la sélection ne porte que sur `lig_produit` (cf question 3.1).

Il faut donc bien choisir l'ordre des colonnes en fonction des requêtes ciblées.

Un seul index sur `(col1, col2)` prendra moins de place et aura un coût de maintenance plus faible que 2 index séparés, i.e. un sur `col1` et un sur `col2`. Il sera plus efficace pour des requêtes dont la sélection porte sur les 2 colonnes, utilisable mais un peu moins efficace si la sélection porte uniquement sur `col1`, pas utilisable si la sélection porte uniquement sur `col2`. Donc, c'est encore une fois les requêtes que l'on cherche à optimiser qui permettront de choisir entre un index multi-colonnes ou deux index simples.

Enfin, il faut éviter un index multi-colonnes si le nombre de colonnes est supérieur à 2.

Le sujet de TP2 fait référence aux index bitmap, pour résoudre la question des combinaisons de critères. Ils sont très pratiques pour combiner des critères de sélection, mais malheureusement ils ne sont pas adaptés à des mises à jours (`insert`, `delete`, `update`) concurrentes. Ils sont donc réservés aux entrepôts de données où les données sont lues et où les mises à jour sont remplacées par de l'alimentation (`insert`) sans concurrence [1].

4 Volumétrie

Le moteur SQL essaie d'évaluer le gain de l'utilisation d'un index, principalement en nombre de blocs lus sur le disque. Il faut compter les blocs de l'index PLUS les blocs de la table. Si cette somme dépasse le nombre total de blocs de la table, autant faire un Scan complet de la table sans lire l'index.

Par exemple, la table `album` est petite, elle tient dans un seul bloc de 8Ko. Donc on perdra toujours du temps à lire l'index puis lire la table, c'est pourquoi le moteur SQL répond à la requête de la question 4.1 sans utiliser l'index.

Autre exemple, dans la question 4.3, si la requête n'est pas assez sélective, ça ne vaut plus le coup d'utiliser l'index parce qu'on va lire beaucoup d'entrées dans l'index (donc de blocs) et beaucoup de lignes de la tables (donc de blocs).

Parfois, on peut répondre à la requête en utilisant uniquement l'index. C'est intéressant parce qu'un index occupe en général moins de blocs que la table indexée, et il est trié.

5 Index partiel

Lié au point précédent, un index partiel permet d'éviter d'indexer les valeurs trop courantes, puisque lorsqu'une sélection porte sur des valeurs très fréquentes (donc peu sélectives), le moteur SQL n'utilise pas d'index.

Un autre cas d'utilisation est d'exclure des valeurs qui n'ont pas d'intérêt. Par exemple, si on stocke des dossiers qui ont une colonne `status` de valeurs possibles `{classé, en_cours}`, on va vouloir indexer uniquement les dossiers `en_cours`, les autres étant beaucoup moins fréquemment consultés.

1. Certaines bases de données proposent des index spatiaux. Il existe de nombreuses propositions d'index à plusieurs dimensions qui tentent d'améliorer l'indexation multi-colonnes.

6 Index couvrant

La création d'un index couvrant permet d'ajouter une ou plusieurs colonne(s) à l'index, sans en modifier la clé, et ainsi de répondre à de nouvelles requêtes avec uniquement cet index. Dans l'exercice 6, l'index `facture_num_include_montant` a pour clé de recherche le numéro de facture, mais contient également le montant de la facture. Un index couvrant peut donc être une bonne solution quand certaines colonnes (ici le montant) sont souvent utilisées en association avec un critère de recherche (ici le numéro de facture). Attention néanmoins à la redondance introduite, l'index couvrant sera utile seulement s'il est le reflet fidèle de la table indexée. De plus, il occupera potentiellement plus de blocs en mémoire du fait de l'ajout des nouvelles colonnes.

7 Bonnes pratiques

Pour optimiser une requête, il faut d'abord se concentrer sur la façon dont on a posé la requête, essayer de la reformuler :

- éviter un `select *` si on n'a pas besoin de toutes les colonnes. Peut-être que le moteur SQL pourra lire uniquement l'index pour répondre à la requête.
- Attention aux conditions de sélection : éviter d'utiliser des fonctions qui "cachent" une colonne indexée
- Eviter des jointures inutiles. Par exemple, si on veut compter le nombre de produits par facture, on n'a pas besoin de la table facture.
- Examiner les plans d'exécution de plusieurs requêtes équivalentes pour choisir celle qui est la moins coûteuse pour votre SGBD. D'un SGBD à l'autre, il peut y avoir des préférences.
- La création d'un nouvel index ne vient qu'en dernier recours, à étudier avec d'autres modifications liées au modèle physique.

8 Pour conclure

Les index sont un élément clef de l'efficacité des SGBD. Ils font l'objet d'une recherche active. Les propositions issues de la recherche sont incluses progressivement dans les SGBD. Il faut ainsi régulièrement regarder les notes qui accompagnent les nouvelles versions et/ou la documentation pour se tenir au courant sur les nouvelles possibilités offertes par les SGBD en terme d'indexation. Des résultats récents (une implémentation est disponible) peuvent changer beaucoup de choses concernant l'efficacité des index (il manque encore une bonne maîtrise de la concurrence).