Séance 10 : Interrogation de données - Optimisation

Plan d'exécution Une requête SQL est déclarative (Elle détermine le Quoi ?). Elle n'indique pas comment calculer le résultat ⇒ programme

Dans un SGBD le programme qui exécute une requête est appelé plan d’exécution. Ce plan d'exécution est un arbre, constitué d’opérateurs, agissant sur des jeux de données.

Source : sys.bdpedia.fr

Choix du plan d'exécution Plan logique Notations Selection : La selection travaille sur R et definit une relation qui ne contient que les tuples de R qui satisfont à la condition (ou prédicat) specifiée.

σ predicat (R) … FROM R WHERE predicat Projection : La projection travaille sur R et definit une relation restreinte à un sous-ensemble des attributs de R, en extrayant les valeurs des attributs specifiés et en supprimant les doublons.

Projection : π a1,…,ak (R) SELECT a1,…,ak FROM R Jointure theta : La theta-jointure définit une relation qui contient les tuples qui satisfont le predicat P du produit cartesien de R et S. Le predicat P est de la forme R.aiθS.bj où θ est l’un des operateurs de comparaison (<,≤,>,≥,=,!=)

θ-join : R ⋈ predicat S FROM R INNER JOIN S on predicat Exemple Requête :

SELECT titre FROM Film f INNER JOIN Role r on r.idFilm=f.id WHERE f.annee=1995 AND r.nom_role='John McClane' Représentation algébrique

Le SGDB et l'optimiseur de requêtes trouve les expressions équivalentes, évalue leur coût et choisit la meilleure. Il est impossible/contre-performant d'énumérer tous les plans possibles (trop long) : on applique des heuristiques. Heuristique classique : réduire au plus tôt la taille des données

en filtrant les nuplets par des sélections en les simplifiant par des projections Illustration de la recherche du plan logique optimal Séparation des sélections :

Application des sélections aux tables concernées :

Principes essentiels :

L’algèbre permet d’obtenir une version opératoire de la requête. Les équivalences algébriques permettent d’explorer un ensemble de plans. L’optimiseur évalue le coût de chaque plan. De manière

Heuristique : l'optimiseur ne peut pas tout explorer Incomplète : Reste à l'optimiseur à choisir le bon algorithme pour chaque opération. Plan physique Le plan physique consiste pour l'optimiseur à choisir les opérateurs.

Notion d'itérateur (open(), next(), close() methods)

Bloquants ⇒ Matérialisation Non bloquants ⇒ Pipelinage Opérateurs bloquants Parfois il n’est pas possible d’éviter le calcul complet de l’une des opérations avant de continuer. On est alors en présence d’un opérateur dit bloquant dont le résultat doit être entièrement produit (et matérialisé en cache ou écrit sur disque) avant de démarrer l’opération suivante. Par exemple:

le tri (order by); la recherche d’un maximum ou d’un minimum (max, min); l’élimination des doublons (distinct); le calcul d’une moyenne ou d’une somme (sum, avg); un partitionnement (group by); On distingue donc :

le temps de réponse: c’est le temps mis pour obtenir un premier nuplet; il est quasi-instantané pour une exécution sans opérateur bloquant; le temps d’exécution: c’est le temps mis pour obtenir l’ensemble du

résultat. Index Un index est une structure de données permettant d'accélérer les recherches dans une table en associant à une clé d'index (la liste des attributs indexés) l'emplacement physique de l'enregistrement sur le disque.

Les accès effectués sur un index se font sur des structures optimisées pour la recherche (liste triée, B-tree…) au lieu de se faire par parcours séquentiel et intégral des enregistrements.

B-Tree Intérêt :

R = Nombre de valeurs possibles/Nombre d'enregistrements ⇒ 1

Les index B-TREE sont les plus couramment utilisés.

Ils doivent être utilisés sur les tables qui sont fréquemment soumises à des recherches et sont d'autant plus pertinents que les requêtes sélectionnent un petit nombre d'enregistrements (moins de 25% par exemple).

Les index doivent être utilisés sur les attributs :

souvent mobilisés dans une restriction (donc une jointure) très discriminés (c'est à dire pour lesquels peu d'enregistrements ont les mêmes valeurs) rarement modifiés Inconvénients :

diminuent les performances en mise à jour (puisqu'il faut mettre à jour les index en même temps que les données). ajoutent du volume à la base de données et leur volume peut devenir non négligeable. Ne permettent pas toujours de gagner en efficacité (voir plus bas). Source : use-the-index-luke.com

Bitmap Intérêt :

R = Nombre de valeurs possibles/Nombre d'enregistrements ⇒ 0

Les index Bitmap sont destinés à l'indexation de colonnes qui comportent donc peu de valeurs distinctes et beaucoup d'enregistrements pour chacune de ces valeurs.

A l'inverse des index B-Tree, les index Bitmap ne stockent pas un pointeur vers un enregistrement dans un fichier trié sur l'index, mais une valeur codée sur un bit (vrai ou faux) pour chaque valeur de la colonne indexée (2 bits pour une cardinalité 2, 3 pour une cardinalité 3, etc.) dans un fichier trié sur la clé. Ils sont donc moins coûteux en espace disque.

De tels index optimisent la recherche relative à une question du type : “l'enregistrement appartient-il à un sous ensemble du domaine sur l'index ?”.

Inconvénients :

diminuent de manière importante les performances en mise à jour (puisqu'il faut mettre à jour les index en même temps que les données). utilisables uniquement pour peu de valeurs distinctes = le meilleur ratio de valeurs distinctes au nombre total occurrences est d'environ 1 pour 1000, FullText Les index FullText permettent de faire des recherches sur les champs de type string (CHAR, VARCHAR et TEXT).

Création :

CREATE FULLTEXT INDEX ind_full_titre ON Livre (titre); Utilisation :

SELECT * FROM Livre WHERE MATCH(titre) AGAINST ('story'); Modes possibles :

IN NATURAL LANGUAGE MODE IN BOOLEAN MODE WITH QUERY EXPANSION Index partiels Permettent de poser un index de manière conditionnelle ⇒ limite la taille de l'index et accélère les recherches :

CREATE INDEX nr_mail

```
        ON mails (subject)
```

```
    WHERE status = 'NR'
```

Efficacité des index L'utilisation d'un index ne suffit pas à la performance, même si la recherche dans l'index est plus rapide qu'un parcours séquentiel :

Une recherche nécessite les 3 opérations suivantes

le parcours de l'arbre la suite de la chaîne de nœuds feuilles la récupération des données de la table L'index n'optimise que le parcours de l'arbre/séquentiel.

Points importants :

Nombre de valeurs (unique ou multiple/nombre de valeurs distinctes) Index composites et importance de l'ordre des champs dans l'index (voir Index composites) Index lents ⇒ un TABLE ACCESS FULL (lecture de plusieurs enreg par bloc) peut être plus rapide qu'un INDEX RANGE SCAN (lecture 1 à 1 de plusieurs enregs) (voir Index lents) Index et usage de fonctions ⇒ index avec fonctions possibles pour fonctions pures (voir Usage de fonctions) Index et requêtes préparées Analyse de requêtes La plupart des SGDB possédent un analyseur de requêtes permettant de visualiser le plan d'exécution d'une requête.

PostgreSQL explain :

explain ⇒ Donne le plan d'exécution estimé explain analyze ⇒ Donne le plan d'exécution réel (attention, exécute la requête) Ressources Explain beautifier Explain documentation Application Benchmark avec et sans Index BTree + Analyse des plans de requête (PostGreSQL) sur :

Select monotable sur champ Join sur champ Select avec fonction (LCASE(name) par exemple) Utilisation de fonction de regroupement Order By Références Cours de Philippe RIGAUX (CNAM) Use the index Luke MySQL performance optimization PostgreSQL performance tuning

From:
http://slamwiki2.kobject.net/ - **SlamWiki 2.1**

Permanent link:
http://slamwiki2.kobject.net/cnam/nfp107/seance10?rev=1682170725

Last update: **2023/04/22 15:38**