Séance 8 : Organisation physique

Valorisation des primitives

Rappel: primitives de base :

Primitive	
#T	Accès à un tuple d'une table relationnelle par accés sur clé primaire, connaissant une valeur de cette clé
?T	Accès à un tuple d'une table relationnelle par accès sur un autre attribut que la clé primaire, ou sélection/restriction de tuples d'une table selon une qualification
T+	Ajout d'un tuple à une table relationnelle
T-	Suppression d'un tuple à une table relationnelle. Cette primitive est notée T- nom de la table.
Tm	Modification d'un tuple d'une table relationnelle, par la modification de valeurs d'un ou plusieurs attributs de ce tuple
TXT'	Jointure entre deux tables relationnelles T et T', par la modification de valeurs d'un ou plusieurs attributs de ce tuple. Cette primitive est notée TXT' noms des tables concernées .

Il est nécessaire de définit le poid relatif entre les différentes primitives :

Conventions pour la valorisation des primitives logiques

1. unité d'accés logique = #T, toutes les autres primitives seront exprimées en fonction de #T.
2. le nombre de tuples occupés par une table T sera noté `NbT(T)`
3. le nombre de pages occupées par une table T sera noté `NbP(T)`

avec :

- NbP(T) = NbT(T)/nombre moyen de tuples dans la page
- le nombre moyen de tuples dans la page dépendra :
- de la taille de la page, noté P
    - de la taille du tuple, noté T
    - du taux de remplissage de la page

ces valeurs seront liées au SGBD et à l'organisation de la table adoptés. on pourra faire des calculs en première approximation avec un taux de remplissage de 80% et des pages de 4K.

Exemple :

- NbT(T) = 10000; P = 4Koctets; taux de remplissage de 80%; T = 200 octets;
- Nombre moyen de tuples par page = 4000×0,8/200 = 16
- NbP(T) = 10000/16 = 625 pages.

Primitive	**Valorisation**	**Détails**
#T	1	sélection selon clé primaire = unité d'accés logique
?T	NbP(T)	
?T	S x (NbP(T) + NbP(I))	si le critère de sélection comprend plusieurs termes dont un faisant intervenir un index secondaire I avec S = facteur de sélectivité du critère, cad la probabilité qu'un tuple satisfasse le critère NbP(T) = nb pages de la table T NbP(I) = nb pages de l'index I
T+ T-	4	le tuple est atteint par 1 accès à l'index + 1 accès à la page contenant le tuple cherché + 1 mise à jour d'une page + 1 mise à jour de l'index.

Primitive	Valorisation	Détails
Tm	3	le tuple est atteint par 1 accès à l'index + 1 accès à la page contenant le tuple cherché + 1 écriture sur page.
TXT'	S x (NbP(T) + (N(T) x NbP(T')))	jointure, avec NbP(T) = nombre page de la table T S = facteur de sélectivité du critère sur T N(T) = nombre de tuples de T satisfaisant le critère NbAP(T') = nombre page de la table T'

Organisation du stockage

L'organisation du stockage choisie au sein du SGDB a une influence sur la recherche :

Pour certaines requêtes, une organisation sera plus performante que d'autres :

mais ce choix peut avoir des conséquences sur :

- Les performances sur d'autres opérations
- Le volume du stockage (espace disque)
- Le maintien de l'intégrité des données (ACID) ⇒ Atomicité, Cohérence, Isolation, Durabilité ⇒ principes fondateurs des transactions
- Les accès concurrents

Moteurs de stockage

Non ordonné

Le stockage non ordonné stocke généralement les enregistrements dans l'ordre où ils sont insérés. Ce type de stockage offre une bonne efficacité d'insertion O(1), mais des temps de récupération inefficaces O(n).

Toutefois, ces temps de récupération sont généralement meilleurs, car la plupart des bases de données utilisent des index sur les clés primaires, ce qui se traduit par des temps de récupération de style O(log n) ou O(1) pour les clés qui sont identiques aux décalages des lignes de la base de données dans le système de stockage.

Ordonné

Le stockage ordonné stocke généralement les enregistrements dans l'ordre et peut devoir les réorganiser ou augmenter la taille du fichier lorsqu'un nouvel enregistrement est inséré, ce qui réduit l'efficacité de l'insertion. Cependant, le stockage ordonné permet une extraction plus efficace, car les enregistrements sont pré-triés, ce qui se traduit par une complexité de O(log n).

Structures

Heap

Les fichiers heap sont des listes d'enregistrements non ordonnées de taille variable. Bien qu'ils partagent un nom similaire, les fichiers heap sont très différents des Heap Memory. Les Heap memory sont ordonnées, contrairement aux fichiers heap.

- Principes
    - Les nouveaux enregistrements étant ajoutés à la fin du fichier, ce qui donne un ordre chronologique

- Récupération efficace lorsque l'identifiant de la mémoire est l'adresse de la mémoire.
- recherche inefficace, car la recherche doit être linéaire.
- la suppression est accomplie en marquant les enregistrements sélectionnés comme “supprimés”.
- nécessite une réorganisation périodique si le fichier est très volatile (modifié fréquemment).



- Avantages
  - efficace pour le chargement de données en masse
  - efficace pour les relations relativement petites car les frais d'indexation sont évités
  - Efficace lorsque les recherches concernent une grande partie des enregistrements stockés.
- Inconvénients
  - pas efficace pour la recherche sélective à l'aide de valeurs clés, surtout si le fichier est important
  - le tri peut prendre beaucoup de temps
  - ne convient pas aux données avec beaucoup de changements

Hash

Les fonctions de hachage calculent l'adresse de la page dans laquelle l'enregistrement doit être stocké en fonction d'un ou plusieurs champs de l'enregistrement. Elles sont choisies de manière à ce que les adresses soient réparties uniformément dans l'espace d'adressage.

l'adresse unique n'étant pas garantie, des mécanismes de détection et de résolution des collisions sont nécessaires.



Avantages et inconvénients

- efficace pour les correspondances exactes sur le champ clé
- ne convient pas à l'extraction de plages, qui nécessite un stockage séquentiel
- calcule l'endroit où l'enregistrement est stocké en fonction des champs de l'enregistrement.
- les fonctions de hachage garantissent une répartition uniforme des données
- des collisions sont possibles, d'où la nécessité de détecter et de restaurer les collisions.

B-Tree

Ce sont les plus couramment utilisés dans la pratique.

- Le temps nécessaire pour accéder à n'importe quel enregistrement est le même car le même nombre de nœuds est recherché.
- L'index est un index complet, il n'est donc pas nécessaire d'ordonner le fichier de données.



Avantages et inconvénients

- structure de données polyvalente - accès séquentiel ou aléatoire
- l'accès est rapide
- supporte efficacement les correspondances exactes, par plage, par clé partielle et par motif.
- les fichiers volatiles sont traités efficacement car l'index est dynamique - il s'étend et se

contracte au fur et à mesure que la table s'agrandit et se réduit.
- moins bien adapté aux fichiers relativement stables - dans ce cas, ISAM est plus efficace.

TODO image

- Fonctionnement des B-Tree
- Visualisation



Implémenter B-TREE en java pour les opérations suivantes :

- Recherche
- Insertion
- Suppression

Vous avez 15 minutes...

voir https://www.programiz.com/dsa/b-tree

Variante B+Tree :

Parcours séquentiel + recherche indexée

- Fonctionnement des B+Tree
- Visualisation

ISAM

ISAM (acronyme de l'anglais Indexed Sequential Access Method)

Organisation séquentielle indexée : Permet un accès séquentiel ou direct (par index)

ISAM, comme les fichiers séquentiels physiques, stocke les enregistrements les uns à la suite des autres, ce qui permet une utilisation très efficace de l'espace disque. Mais contrairement aux fichiers séquentiels physiques, les données ISAM doivent être stockés sur le disque, car les adresses du disque sont nécessaires pour créer les index. L'emplacement physique des enregistrements dans ISAM n'est pas important puisque les index se chargent de l'accès aux enregistrements. Un seul fichier ISAM peut comporter plusieurs dizaines d'index, chacun permettant de retrouver les fichiers dans un ordre prédéfini. Dans certains cas, la taille des index dépassera la taille du fichier de base. MyISAM utilise B-Tree

- Mysql ⇒ MyISAM
- PostGreSQL ⇒ Unlogged tables
- Oracle ⇒ Tablespaces

MyISAM ne gère ni les clés étrangères, ni les transactions. Il n'y a pas à vérifier la validité des enregistrements. Cela permet donc un précieux gain de temps sur des tables très fréquemment ouvertes en écriture/lecture.

Lors de suppressions sur des champs de type VARCHAR, CHAR, BLOB ou TEXT, le moteur supprime le contenu mais la place précédemment supprimée est conservée et peut être réutilisée ultérieurement. L'utilisation de **OPTMIZE** peut défragmenter la table afin de gagner de la place et ainsi faciliter l'accès aux données.

MyISAM ⇒ très utilisé pour le Web vitrine.

Une table MyISAM utilise trois fichiers :

- maTable.FRM : Fichier de définition de la table
- maTable.MYD : Fichier contenant les données de la table
- maTable.MYI : Fichier d'index

Permet les recherches en FULL-TEXT ⇒ Recherche avec retour de pertinance

Exemple : <sxh sql;gutter:false SELECT *, MATCH (Titre, Article) AGAINST ('base de données') AS Score FROM Article ORDER BY score DESC; </sxh> **Retour :**

```
IdArticle  | Titre     | Article                                          | Score
7          | Federated | Le moteur Federated permet de déporter…          |
0,21985759722453
6          | Exemple   | Ce type de table est assez particulier…          |
0,15944281721118
8          | InnoDB    | C'est le moteur transactionnel le plus utilisé…  |
0,14023887676722
1          | MyIsam    | Ce moteur est une version évoluée d'ISAM avec…   |
0,068685997386924
2          | Memory    | Les tables de type Memory stockent les…          | 0
```



- Avantages
  - Moteur rapide.
  - Possibilité d'écrire et lire en même temps sans risque de verrouillage de table.
  - Verrouillage de table manuel.
  - La mise en cache des clés.
  - Gain de place sur le disque.
  - Gain de mémoire lors des mises à jour.
  - Gestion de la recherche FULL-TEXT.
- Inconvénients
  - Pas de gestion des contraintes de clés étrangères
  - Pas de gestion des transactions (pas de COMMIT / ROLLBACK possible).

InnoDB

InnoDB est le moteur par défaut à partir de MySQL 5.5.52. Son principal avantage par rapport aux autres moteurs de stockage de MySQL est qu'il permet des transactions ACID (Atomiques, Cohérentes, Isolées et Durables), ainsi que la gestion des clés étrangères (avec vérification de la cohérence). Autrement dit, InnoDB est un moteur de bases de données relationnelles et transactionnelles, comparable à celui utilisé par PostgreSQL.

Toutes les bases de données sont stockées au même endroit. Par défaut dans le fichier ibdata qui se trouve généralement dans le dossier /var/lib/mysql/ . Il est également possible d'utiliser plusieurs fichiers ou même d'utiliser directement une ou plusieurs partitions sur le disque en mode RAW.

Les fichiers de définitions de table .frm sont également dans un dossier au nom de la base comme pour MyISAM.

Les fichiers données de InnoDB ne peuvent pas être sauvegardés par copie de fichiers : il en résulterait une corruption de données. La seule façon de faire une copie des fichiers à chaud est d'utiliser Percona XtrabBackup pour MySQL et Mariabackup pour MariaDB, ou d'utiliser la commande mysqldump en mode “ single transaction”.

Depuis la version 5.6, une extension memcached peut être associée à InnoDB, permettant d'améliorer les performances.



- Avantages
  - Verrouillage de ligne.
  - Gestion du COMMIT/ROLLBACK
  - Gestion de gros volumes de données.
  - Gestion des clés étrangères.
  - Grande panoplie d'éléments de configuration du moteur.
  - Gestion du backup sans bloquer une base en production.
  - Couramment disponible chez les hébergeurs en mutualisé.
- Inconvénients
  - Lenteur de certaines opérations telles que le SELECT COUNT(*) FROM maTable.
  - Statistiques envoyées ne sont pas forcément précises : ce ne sont que des estimations.

Choix d'organisation

séquentiel (HEAP)

Conditions favorables :

- chargement de données dans la base
- petites tables (occupation de peu de pages)
- requêtes manipulant des tables entières
- pour recupérer de l'espace dû à des DELETE

Eviter :

- accès à 1 ou plusieurs tuples
- grosses tables

HASH

Conditions favorables :

- recherche sur valeur exacte de clé (le plus rapide)

Eviter :

- recherche sur pattern matching (partie de clé)
- traitement de table entière
- joints naturels (systématique sans restriction)

ISAM

Conditions favorables :

- requêtes nécessitant pattern matching ++
- la table grossit lentement (peu de réorg.)

- clé large

Eviter :

- si recherche sur clé complète (→HASH)
- grosse table à croissance rapide

BTREE

Conditions favorables :

- besoin de pattern matching +
- la table grossit vite
- table trop grosse pour être souvent réorganisée (Modify)
- joints de tables entières

Eviter :

- table statique ou à croissance faible
- large clé
- si ajout de nouveaux tuples seulement en fin de table (plus grand risque de DEAD LOCK)

Résumé

Fonctionnalité	B-Tree	ISAM	Hash	Heap
chargement de table	++	–	–	+
recherche sur clé complète	++	++	+	–
intervalles/pattern matching	+	+	–	–
recherches séquentielles	++	++	-	+
recherche sur clé partielle	+	+	–	–
accés à données triées	+	–	–	–
joints sur larges tables	+	+	–	–
index croît comme table	+	–	–	–
très petite table	+	–	–	–
très grande table	+	–	–	–

Source : Bernard Espinasse

Application

Objectif :

- Benchmark des moteurs de stockage MySQL (InnoDB, MyISAM) et PostgreSQL.
- Tests sur petits (100) et + gros volumes de données (10 000)
- sur les 6 primitives de base



- Etablir un protocole de test (nb de requêtes, concurrency level, méthode/outils utilisés)
- Créer les bases de données :
  - Pour chaque moteur (X3)
  - Pour chaque volume (X2)
- Incorporer les données avec GenerateData
- Mettre en place le protocole établi



- Réaliser les tests
- Présenter les résultats

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

Permanent link:
http://slamwiki2.kobject.net/cnam/nfp107/seance8

Last update: **2023/04/22 16:20**