Bases de données
Optimisation – 1 : introduction
Dénormalisation - Indexation – Hachage - Moteurs

Bertrand LIAUDET

SOMMAIRE

Edition juillet 2015

INTRODUCTION

Bibliographie

Gardarin. Bases de données. Eyrolles 2005. §10 – pp. 301 à 350.

Darmaillac, Rigaux. Maîtriser MySQL 5. O'Reilly 2005. §11 – pp. 283 à 318.

Dubois, Hinz, Pedersen. MySQL-Guide officiel. Campus press 2004. §13 – pp. 433 à 475.

MySQL 5-Guide de l'administrateur. Campus press 2005. §6 – pp. 447 à 510.

Harrison. MySQL Stored Procedure. O'Reilly 2006. Part IV – pp. 421 à 582.

http://dev.mysql.com/doc/refman/5.0/fr/mysql-optimization.html

Navaro. Optimisation des bases de données : mise en œuvre sous Oracle. Pearson. 2010. A noter que ce livre traite aussi de MySQL et SQL server.

Généralités sur l'optimisation

Objectif de l'optimisation

L'objectif de l'optimisation est d'accélérer la vitesse de traitement des requêtes pour rendre le système plus satisfaisant du point de vue de l'utilisateur.

La question de la mémoire

Caractéristiques de la mémoire

- Réinscriptibilité : l'information peut être modifiée ou pas.
- Durabilité : l'information est conservée ou pas.
- Rapidité : rapidité d'accès à la mémoire. C'est la principale question économique.
- Taille :

Situation en 2005

En fonction des 3 caractéristiques précédentes, on a les différentes mémoires existantes :

Réinscriptibilité	Durabilité	Rapidité	Taille en MO	Mémoires existantes
Non réinscriptible	Non durable			∅
	Durable	1 s	Quelques milliers	CD, DVD, etc.
Réinscriptible	Non durable	10^{-8} s	Quelques centaines (+)	Mémoire cache
		10^{-8} à 10^{-7} s	Quelques milliers	Mémoire vive (principale)
	Durable	10^{-2} s	Quelques millions	Mémoire secondaire (disque dur)

		1 seconde	Jusqu'à quelques milliards	Mémoire tertiaire (bande, CD, DVD)

A bien noter :

La différence de rapport de rapidité entre la mémoire cache et la mémoire secondaire (disque dur) est de l'ordre de 1 million !

De ce fait, **ce sont essentiellement les accès disques qui seront coûteux** lors des algorithmes, bien plus que les calculs.

La question de la performance : l'évolution des SGBD en terme de rapidité

Capacité de traitement

- **Années 70 :** quelques requêtes par seconde
- **Aujourd'hui :** plusieurs milliers de transactions par seconde

4 causes à l'amélioration des performances des SGBD

- Augmentation de la vitesse du processeur
- Amélioration de la production des plans d'exécution et le leur choix
- **Optimisation des méthodes d'accès aux données (les algorithmes de calcul)**
- **Utilisation de mémoire cache dans les méthodes d'accès aux données**

Temps d'entrée-sortie : 10^{-2} seconde

A noter que les temps d'entrée-sortie disque restent à peu près constant : de l'ordre de la **dizaine de millisecondes,** d'où l'intérêt de les limiter par l'optimisation.

Les différentes approches de l'optimisation

L'optimisation concerne finalement la question des accès disques.

Cette question peut s'aborder de différents point de vue :

- Hardware
- Software, côté serveur
- Software, côté client

Optimisation Hardware : administration système

L'optimisation hardware au sens large concerne la **puissance de la machine**, la **mémoire de la machine**, la **fluidité du réseau**, **l'architecture client-serveur globale**.

Par exemple : l'utilisation de RAID-RAID 0 (gestion des disques) est préconisée pour l'optimisation des opérations de lecture écriture. Ainsi que l'utilisation des disques SCSI plutôt que les IDE.

Software, côté serveur : administration SGBD

L'optimisation software côté serveur concerne le **paramétrage du serveur** pour améliorer l'efficacité des traitements. Il s'agit essentiellement de **paramétrer des mémoires caches** pour mieux contrôler les accès disques. Ca relève de ce qu'on appelle le « **tuning** » du serveur ou de la BD. C'est très spécifique selon les SGBD.

Exemple de paramètres système MySQL

key_buffer_size: mémoire utilisé pour la sauvegardes de indexes MyISAM.

table_cache: Nombre de tables ouvrables simultanément.

read_buffer_size: mémoire utilisée pour la sauvegarde des données issues de select complet de tables.

sort_buffer: mémoire utilisée pour la sauvegarde des données de tables qui seront triées par un ORDER BY.

Etc.

Software, côté application : développement

L'optimisation software côté application concerne :

L'optimisation du DDL

Il s'agit de produire un modèle relationnel optimisé (niveau MPD dans la méthode MERISE).

Cette optimisation concerne :

- **La dénormalisation**

Elle permet de limiter les jointures donc d'accélérer certains traitements ou de gagner de la place. En contre partie, la dénormalisation génère des risques de production de données incohérentes.

- **L'indexation**

Elle permet de limiter le temps de recherche donc d'accélérer certains traitements.

Attention, trop d'index ralentit aussi les traitements !

- **Limiter la gestion des droits**

Les privilèges au niveau des tables et des colonnes ralentissent les traitements. Supprimer ces privilèges permet d'optimiser les traitements au pris d'une délégation du contrôle des droits au niveau du client.

- **Contrôler la taille des types**

Essayer autant que possible de réduire la taille des types : TINYINT à la place de INT, char(10) à la place de char(20), etc.

- **Eviter les types de taille variable**

Il vaut mieux éviter les types de tailles variables (VARCHAR, BLOB et TEXT) qui compliquent l'organisation en mémoire, surtout pour les tables qui changent beaucoup.

- **Gestion des NULL**

Eviter les NULL. Utiliser le « NOT NULL autant que possible » et utiliser la valeur « 0 » à la place d'une valeur NULL si possible.

- **Gestion des valeurs par défaut**

Profiter des valeurs par défaut pour éviter de préciser la valeur.

L'optimisation des requêtes

Il s'agit ici d'optimiser la requête pour la rendre plus efficace. Cela concerne le bon usage de la commande EXPLAIN.

Le repérages des requêtes lentes peut se faire à l'observation du temps de réponse, en faisant des benchmarks ou en observant le log des requêtes lentes. Il est possible d'activer cette option au niveau de MySQL paramétrant la variable --log-slow-queries

L'optimisation MySQL

- Eviter les SELECT trop complexes sur les tables fréquemment mises à jour.
- Faire des OPTIMIZE TABLE après avoir supprimé un grand nombre de lignes.

http://dev.mysql.com/doc/refman/5.0/fr/mysql-optimization.html

DENORMALISATION

1. Présentation

Principe

La dénormalisation est une optimisation en vue de :

- accélérer les traitements de consultation pour améliorer les temps de réponse pour l'utilisateur.
- réduire la taille des tables

La dénormalisation consiste à « casser » le modèle relationnel normalisé des données.

Coût

La dénormalisation a toujours un coût :

- Soit en terme de sécurité : l'intégrité des données est moins bien garantie
- Soit en terme de performance : ce qu'on gagne en consultation (SELECT), on le perdra en DML (insert, update, delete).

Justification

La dénormalisation, comme toute optimisation de la BD, doit être justifiée par l'usage réel du système : étant donné les machines utilisées, le nombre de tuples dans la BD, le nombre d'utilisateurs du système, la dénormalisation peut se justifier.

La dénormalisation ne doit jamais être justifiée par l'intérêt du programmeur !

Attributs calculés

On appelle attribut calculé un attribut automatique dont la valeur est le résultat d'un calcul fait à partir de données présentes dans la BD.

Un tel attribut présente le défaut de dupliquer une information déjà présente dans la BD.

Les attributs calculés peuvent être gérés par des vues : leur réalité n'est donc que virtuelle.

Ils peuvent aussi être gérés en créant réellement un nouvel attribut. Il faudra alors vérifier la cohérence des données avec des triggers.

L'utilisation d'attributs calculés est le cas le plus courant d'optimisation.

Exemple

Version normalisée

Disques(ND, titre, dateSortie, maisonDisque)

Chansons(NC, titre, durée, dateCréation)

Composer(#ND, #NC)

Un disque est composé de plusieurs chansons. Une chanson peut appartenir à plusieurs disques.

Version dénormalisée

Disques(ND, titre, dateSortie, maisonDisque, ***duréeDisque)***

Chansons(NC, titre, durée, dateCréation)

Composer(#ND, #NC)

« duréeDisque » est fonction de la durée de chaque chanson du disque. Il faudra le gérer avec un trigger ;

Version avec attribut calculé géré par une vue

```
Create view DisquesAvecDurée as
Select D.ND, D.titre, .D.dateSortie, D.maisonDisque, count(C.durée) as duréeDisque
From Disques D, Chansons C, Composer CO
Where CO.ND=D.ND
And CO.NC=C.NC
Group by D.ND, D.titre, D.dateSortie, D.maisonDisque
```

Une telle vue simule la présence de l'attribut calculé « duréeDisque » : cependant, il faut recalculer à chaque fois la valeur de la durée du disque.

Transitivité des DF : ajout de liens directs

Un lien direct est une clé étrangère qui relie deux tables qui sont par ailleurs reliées par une ou plusieurs clés étrangères passant par une ou plusieurs tables intermédiaires.

L'intérêt de ce lien est de limiter le nombre de jointures lors des requêtes.

Il faudra vérifier la cohérence des données avec des triggers.

Le lien direct peut être considéré comme un attribut calculé.

Exemple

Version normalisée

Employés(NE, nom, dateEmbauche, salaire, #NB)

Bureaux(NB, étage, surface, #ND)

Départements(ND, nomDept, villeDept)

Version dénormalisée

Employés(NE, nom, dateEmbauche, salaire, #NB, #ND)

Bureaux(NB, étage, surface, #ND)

Départements(ND, nomDept, villeDept)

La table Employés n'est pas en 3ème forme normale car NB -> ND.

Fusion de tables

La fusion des tables consiste à violer les formes normales 2 ou 3.

L'intérêt de cette fusion est d'éviter d'avoir à faire des jointures lors des requêtes.

Il faudra vérifier la cohérence des données avec des triggers.

Exemple

Version normalisée :

Employés(NE, nom, dateEmbauche, salaire, #ND)

Departements(ND, nomDept, villeDept)

Version dénormalisée

Employés(NE, nom, dateEmbauche, salaire, ND, nomDept, villeDept)

Cette table n'est pas en 3ème forme normale car ND -> nomDept, villeDept.

Conséquences de la version dénormalisée

Pour garantir l'intégrité des données, il faudra compenser l'absence de clé étrangère par le codage d'un trigger qui vérifie, pour chaque insertion d'un nouveau tuple, que pour un ND donné, on a bien un seul nomDept et un seul villeDept.

Cette optimisation se justifie si les jointures entre les deux tables sont coûteuses et si l'application du trigger est moins coûteuse, coûteux voulant dire : pénalisant pour l'utilisateur.

Duplication d'attributs

La duplication d'attributs consiste à violer les formes normales 2 ou 3.

C'est une forme réduite de la fusion des tables : on ne fusionne qu'une partie de la table.

L'intérêt de cette duplication est d'éviter d'avoir à faire des jointures lors des requêtes.

Il faudra vérifier la cohérence des données avec des triggers.

Exemple 1

Version normalisée

Employés(NE, nom, dateEmbauche, salaire, #ND)

Departements(ND, nomDept, villeDept)

Version dénormalisée

Employés(NE, nom, dateEmbauche, salaire, ND, nomDept)

Departements(ND, nomDept, villeDept)

La table Employés n'est pas en 3ème forme normale car ND -> nomDept

Conséquences de la version dénormalisée

Pour garantir l'intégrité des données, il faudra compenser l'absence de clé étrangère par le codage d'un trigger qui vérifie, pour chaque insertion d'un nouveau tuple dans « employés », viendra affecter la valeur de nomDept trouvée dans la table « Départements ». Si la valeur de nomDept est proposée, il faudra vérifier qu'elle est cohérente avec celle de la table « Départements ». Si le ND proposé n'existe pas, on pourra créer un nouveau département en plus du nouvel employé.

Cette optimisation se justifie si les jointures entre les deux tables sont coûteuses et si l'application du trigger est moins coûteuse, coûteux voulant dire : pénalisant pour l'utilisateur.

Exemple 2

Version normalisée

Factures(NF, dateFacture, montantFacture)

Règlements(NR, dateRèglement, montantRèglement, #NF)

Un règlement concerne une facture et une seule.

Une facture donne lieu à 0 ou 1 règlement.

Version dénormalisée

Factures(NF, dateFacture, montantFacture, #NR)

Règlements(NR, dateRèglement, montantRèglement, #NF)

Gestion d'un attribut solitaire dans une table

Exemple

Version normalisée

Employés(NE, nom, dateEmbauche, fonction, salaire)

Version dénormalisée

Employés(NE, nom, dateEmbauche, salaire, #NF)

Fonctions(NF, fonction)

Ou

Employés(NE, nom, dateEmbauche, salaire, # fonction)

Fonctions(fonction)

Principe

Quand on a un attribut dont l'extension (liste de valeurs possibles) est faible par rapport aux nombres de tuples portant cet attribut,

Ou si on souhaite pouvoir contrôler plus finement cette extension,

Alors on peut avoir intérêt à créer une table spécifique pour cet attribut.

INDEXATION

1. Problématique

Le problème de l'indexation est celui de l'accès aux données : comment rechercher rapidement à une donnée dans la BD (travail du SGBD) et dans un disque dur en général (travail du SE).

2. Recherche et ajout d'un élément : approche intuitive

On a une collection d'éléments : comment chercher et ajouter un élément dans cette collection ?

Il existe plusieurs méthodes associées à une organisation particulière des données

Avec un tableau quelconque

Méthode

On parcourt tous les éléments du tableau jusqu'à ce qu'on trouve celui cherché.

Complexité de recherche

O(n/2)

Complexité d'ajout

O(n/2)

Inconvénients

C'est long et la durée est aléatoire (fonction de la place de l'élément cherché).

Avec un tableau trié

Méthode

On utilise la méthode de la recherche dichotomique.

Complexité de recherche

O(log2(n)

Complexité d'ajout

O(n/2)

Avantages

On trouve l'élément en Log2(n) tests au maximum (par exemple, avec 1 000 000 éléments, soit environ 2 puissance 20 éléments, on trouve l'élément cherché en maximum 20 tests).

Inconvénients

Le maintient du tri en cas d'ajout ou de suppression d'un élément. On est obligé de déplacer tous les éléments en dessous de celui qu'on ajoute ou qu'on supprime. C'est d'autant plus long que les éléments du tableau sont volumineux.

Avec un tableau trié de pointeurs

Méthode

Recherche dichotomique avec un tableau de pointeurs

Complexité de recherche

O(log2(n)

Complexité d'ajout

O(n/2)

Avantages

Ceux de la recherche dichotomique.

Inconvénients

Ceux de la recherche dichotomique.

Toutefois, par rapport à un tableau d'éléments, on a seulement une série de pointeurs à déplacer.

Avec une liste chaînée triée

Méthode

On parcourt tous les éléments de la chaîne jusqu'à ce qu'on trouve celui cherché.

Complexité de recherche

O(n/2)

Complexité d'ajout

O(n/2)

Avantages

On peut insérer facilement un élément

Inconvénients

C'est long et la durée est aléatoire (fonction de la place de l'élément cherché).

Avec un arbre binaire

Méthode

Recherche dichotomique avec un arbre binaire.

Complexité de recherche

O(log2(n)

Complexité d'ajout

O(log2(n)

Avantages

On peut insérer facilement un élément

Si l'arbre est équilibré, le nombre de tests est celui d'une recherche dichotomique.

Inconvénients

Le maintien de l'équilibre de l'arbre est complexe et peut être coûteux.

Bilan général de la recherche : séquentielle et dichotomique

On vient de voir qu'il y a deux grandes méthodes de recherche :

- **la recherche séquentielle sur des données quelconques.**
- la recherche dichotomique sur des données triées.

La recherche dichotomique est la méthode de base pour un accès aux données efficace. L'utilisation d'arbre binaire plutôt que de tableau est la base pour des insertions et suppression de données efficaces.

3. Indexation et BD relationnelle

Présentation

Dans les BD relationnelles, l'indexation est une technique qui va permettre d'accélérer les données en permettant de faire des recherches dichotomiques.

On peut se représenter un index comme étant un tableau à deux attributs : le premier est l'attribut indexé. Le second est l'adresse du tuple correspondant dans la BD. Un index est un tableau trié selon l'attribut indexé.

Exemple

Soit le table des employés :

Adresse	NE	Nom	Job	DateEmb	Salaire	Comm	#ND	*NEchef
Ad1	7839	KING	PRESIDENT	17/11/81	5000	NULL	10	NULL
Ad2	7698	BLAKE	MANAGER	01/05/81	2850	NULL	30	7839
Ad3	7782	CLARK	MANAGER	09/06/81	2450	NULL	10	7839
Ad4	7566	JONES	MANAGER	02/04/81	2975	NULL	20	7839
Ad5	7654	MARTIN	SALESMAN	28/09/81	1250	1400	30	7698
Ad6	7499	ALLEN	SALESMAN	20/02/81	1600	300	30	7698
Ad7	7844	TURNER	SALESMAN	08/09/81	1500	0	30	7698
Ad8	7900	JAMES	CLERK	03/12/81	950	NULL	30	7698
Ad9	7521	WARD	SALESMAN	22/02/81	1250	500	30	7698
Ad10	7902	FORD	ANALYST	03/12/81	3000	NULL	20	7566
Ad11	7369	SMITH	CLERK	17/12/80	800	NULL	20	7902
Ad12	7788	SCOTT	ANALYST	09/12/82	3000	NULL	20	7566
Ad13	7876	ADAMS	CLERK	12/01/83	1100	NULL	20	7788
Ad14	7934	MILLER	CLERK	23/01/82	1300	NULL	10	7782

Indexer la clé primaire et le job consiste à créer les deux tables d'index suivantes :

Adresse	NE
Ad11	7369
Ad6	7499
Ad9	7521
Ad4	7566
Ad5	7654
Ad2	7698
Ad3	7782
Ad12	7788
Ad1	7839
Ad7	7844
Ad13	7876
Ad8	7900
Ad10	7902
Ad14	7934

Adresse	Job
Ad10	ANALYST
Ad12	ANALYST
Ad8	CLERK
Ad11	CLERK
Ad13	CLERK
Ad14	CLERK
Ad2	MANAGER
Ad3	MANAGER
Ad4	MANAGER
Ad1	PRESIDENT
Ad5	SALESMAN
Ad6	SALESMAN
Ad7	SALESMAN
Ad9	SALESMAN

Dans les BD, l'index est toujours trié. Ainsi, on pourra faire une recherche dichotomique à partir de NE ou à partir du job et accéder aux tuples.

Index primaires et index secondaires : unicité ou multiplicité du résultat

Index primaire ou unique

Les index primaires sont ceux qui s'appliquent à la clé primaire ou aux clés secondaires d'une table.

La sélectivité est alors la plus petite possible : 1 / nombre de tuples de la table.

La recherche conduit à un élément et un seul. On veut par exemple l'employé n° 7654. La sélectivité est de 100/14 %

Index secondaire ou non unique

Les index secondaires sont ceux qui s'appliquent à des attributs qui ne sont ni clés primaires ni clés secondaires.

Dans ce cas la sélectivité est supérieure à 1 / nombre de tuples de la table.

La rechercher conduit à plusieurs éléments. On recherche par exemple tous les SALESMAN et on obtient 4 employés. La sélectivité est de 100*4 / 14 % soit 28,5%.

Statistiquement, cette sélectivité ne devrait pas beaucoup bouger : elle correspond au taux de SALESMAN nécessaire au fonctionnement de l'entreprise.

Utilité d'un index : sélectivité < 30 %

Pour qu'une recherche indexée soit efficace, il faut que la sélectivité reste faible.

En pratique, on considère que pour une sélectivité > 20 ou 30%, l'index est inefficace.

Ceci vient du fait que les accès disques sont l'élément le plus pénalisant en terme de coût. Or, l'accès direct à un grand nombre N d'éléments conduit à au même nombre N d'accès disque, tandis qu'en accès séquentiel, le nombre d'accès disque est limité du fait de la lecture du disque par page et de la présence de plusieurs tuples sur la même page.

Index plaçant et non plaçant

Un index plaçant est un index dont le tri correspond au tri physique sur le fichier.

En général, la clé primaire est un index plaçant. D'où l'intérêt de ne pas gérer soi-même la clé primaire et d'en faire un attribut auto-incrémenté.

Les index plaçant sont en général organisé avec des arbres B+ : l'arbre permet de gérer les index et les tuples de la table.

Index bitmap

L'index bitmap s'applique aux attributs ayant un nombre de valeurs limité.

Il consiste à créer une table contenant comme attributs toutes les valeurs possibles de l'attribut de départ qu'on veut indexer. Dans cet index, les valeurs possibles seront 0 ou 1 selon les attributs de l'index correspondent ou pas à la valeur de l'attribut de départ.

Un index bitmap est donc une matrice creuse (constituée de 0 ou de 1) encore appelée « bitmap ».

Il n'y a pas de tri, mais seulement le remplacement d'une valeur de type chaîne de caractères par une valeur de type booléen.

Le tri n'est pas nécessaire car l'index (0 ou 1) donne directement la valeur sur un bit.

Exemple : index bitmap sur le JOB

Adresse	PRESIDENT	MANAGER	SALESMAN	CLERK	ANALYST
Ad1	1				
Ad2		1			
Ad3		1			
Ad4		1			
Ad5			1		
Ad6			1		
Ad7			1		
Ad8				1	
Ad9			1		
Ad10					1
Ad11				1	
Ad12					1
Ad13				1	
Ad14				1	

4. Indexation MySQL

Index MySQL : INDEX = KEY

Les 4 types d'index MySQL

Index de clé primaire (**unique**) : PRIMARY KEY

Index **unique** (non clé primaire) : UNIQUE

Index **non unique** : INDEX ou KEY (pour les clés étrangères et les attributs de recherche).

Index de texte : FULLTEXT (on utilise ensuite MATCH et AGAINTS pour les recherche).

Création des index

CREATE TABLE ... [...PRIMARY KEY..], [...FOREIGN KEY..], [...KEY...], [..INDEX...]

ou

ALTER TABLE nomTable ADD INDEX (nomAttribut)

ou

CREATE INDEX nomIndex ON nomTable (nomAttributt)

Suppression des index

DROP INDEX nomAttribut ON nomTable;

La commande SHOW INDEX

```
mysql> show index from livres\G
*************************** 1. row ***************************
       Table: livres
  Non_unique: 0
    Key_name: PRIMARY
Seq_in_index: 1
 Column_name: NL
   Collation: A
 Cardinality: 30
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
*************************** 2. row ***************************
       Table: livres
  Non_unique: 1
    Key_name: NO
Seq_in_index: 1
 Column_name: NO
   Collation: A
 Cardinality: 30
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
2 rows in set (0.01 sec)
```

Non unique : precise si l'index est unique ou pas.

Key name : précise le nom de l'index

Seq in index : précise la position de l'attribut dans l'index dans le cas d'index concaténé.

Cardinality : précise le nombre de tuples de la tables.

Mesurer l'effet d'un index : SQL_NO_CACHE

Pour mesurer la différence entre une requête indexée et une requête non indexée, il faut utiliser la clause : SQL_NO_CACHE juste après le select : SELECT SQL_NO_CACHE etc.

Cela permet que la mémoire cache ne soit pas utilisée par le SGBD. Si on n'utilise pas cette clause, en testant deux requêtes identiques on obtient toujours un résultat avantageux pour la deuxième qui utilise les résultats déjà placés dans le cache, limitant ainsi les accès disques.

Index et %, index et inégalité

Quand on utilise un % au début ou au milieu d'une chaîne, l'index est inéficace, voir pénalisant.

Quand on utilise un % en fin de chaine, l'index est efficace.

Quand on utilise inégalité (>, >=, <, <=), l'index sera d'autant plus efficace que le résultat concerne une population résutante plus petite. Par exemple si on écrit: "where nom < 'B', on obient tous les noms en A et le résultat sera obtenu naturellement plus rapidement que si on demande "where nom < 'S'.

Principaux index

La clé primaire

Les clés primaires doivent toujours être indexées pour permettre des jointures efficaces.

Les clés étrangères

Les clés étrangères peuvent être indexées ou pas.

MySQL indexe automatiquement les clés étrangères quand on les déclare (rappelons toutefois que MyISAM ne gère pas l'intégrité référentielle).

Indexer une clé étrangère est utile si une requête statique régulièrement utilisée fait des restrictions sur cet attribut ou pour permettre augmenter les possibilité d'optimisation sur les jointures multiples.

Les autres attributs

Tout attribut peut être indexé.

Indexer un attribut est utile si une requête statique régulièrement utilisée fait des restrictions sur cet attribut.

Rappelons toutefois qu'en pratique, on considère que pour une sélectivité > 20 ou 30%, l'index est inefficace.

Index “Full Text” et Match

Présentation

- ***Définition de la recherche Full-Text***

  Recherche de mots-clés dans des textes.

- ***Solution naïve, sans Full-Text***

  Select NA, titre, texte
  From articles
  Where (text like “%motCle1%” or titre like “%motCle1%”) and
  (text like “%motCle2%” or titre like “%motCle2%”);
  Cette solution n’est pas optimisée.

Index Full-Text MySQL

http://dev.mysql.com/doc/refman/5.0/fr/fulltext-search.html

Création d’un index Full-Text

- ***Contraintes***

  Le Full-Text ne fonctionne qu’en MyISAM.
  La recherche Full-Text est sensible aux accents.

- ***Syntaxe***

```
CREATE TABLE articles (
        NA integer primary key,
        Titre varchar(100),
        Texte text
) ENGINE=MyISAM;

CREATE  FULLTEXT  INDEX  indexMC ON articles (titre, texte);
```

- ***Méthode de fonctionnement***

  A chaque insertion ou modification dans la BD d’un titre ou d’un texte, l’index découpe les titres ou les textes en mots et gère un index des mots-clés.
  Les mots-clés sont découpés selon les séparateurs de mots : espaces, signes de ponctuation.
  L’apostrophe n’est pas un séparateur de mot.
  Un mot a au moins 4 caractères.
  MySQL définit une liste de mots anglais non pertinents (able, about, above, etc.).
  Les mots présents dans plus de la moitié de la population sont trop généraux et éliminés.

- ***Paramétrage fin***

Certains éléments sont paramétrables :

5.0 § 12.6.4 http://dev.mysql.com/doc/refman/5.0/fr/fulltext-fine-tuning.html

5.6 § 12.9.6 http://dev.mysql.com/doc/refman/5.6/en/fulltext-fine-tuning.html

Recherche Full-Text : MATCH … AGAINT

- ***Avec un seul mot-clé***

```
SELECT NA, Titre, Texte
FROM articles
WHERE MATCH (Titre, texte) AGAINT (‘MySQL ‘) ;
```

- ***Avec plusieurs mots-clés : les uns OU les autres***

```
SELECT NA, Titre, Texte
FROM articles
WHERE MATCH (Titre, texte) AGAINT (‘MySQL ORACLE‘) ;
```

On obtiendra les articles ayant comme mots-clés ou “MySQL” ou “ORACLE” (ou les deux).

- ***Avec plusieurs mots-clés : les uns ET les autres***

```
SELECT NA, Titre, Texte
FROM articles
WHERE MATCH (Titre, texte) AGAINT (‘MySQL‘)
and MATCH (Titre, texte) AGAINT (‘ORACLE‘) ;
```

On obtiendra les articles ayant comme mots-clés et “MySQL” et “ORACLE”.

- ***Avec suppression de mots-clés***

```
SELECT NA, Titre, Texte
FROM articles
WHERE MATCH (Titre, texte) AGAINT (‘MySQL ORACLE‘)
And NOT  MATCH (Titre, texte) AGAINT (‘PostgreSQL‘) ;
```

Classement des résultats: calcul de la pertinence

- ***Affichage de la pertinence***

```
SELECT NA, Titre, Texte, MATCH (Titre, texte) AGAINT (‘MySQL ‘) as pertinence
FROM articles
WHERE MATCH (Titre, texte) AGAINT (‘MySQL ‘) ;
```

- ***Calcul de la pertinence***

Le principe du calcul est le suivant :

Un mot-clé présent dans un texte court a plus de poids qu’un mot clé présent dans un texte long.

Un mot-clé présent dans peu de tuples aura plus de poids qu’un mot clé présent dans beaucoup de tuples.

Avec ces principes, on obtient un classement qui correspond à l’ordre d’affichage des résultats.

- ***Sélection des plus pertinents***

  WHERE MATCH (Titre, texte) AGAINT (‘MySQL ‘) ;

  Est équivalent à:

  WHERE MATCH (Titre, texte) AGAINT (‘MySQL ‘) > 0 ;

  On peut prendre uniquement les 5 premiers :

  WHERE MATCH (Titre, texte) AGAINT (‘MySQL ‘) Limits 5;

Full-Text et InnoDB

Le Full-Text ne fonctionne qu’en MyISAM.

On peut utiliser le Full-Text en InnoDB en créant une table MyISAM pour le Full-Text dans la BD InnoDB.

- ***Exemple***

  Articles (NA, texte, autres attributs) InnoDB ;

  Autres tables ( ) InnoDB ;

  Tranformé en :

  ArticleFT (#NA, texte) MyISAM ;

  Articles (NA, autres attributs) InnoDB ;

  Autres tables ( ) InnoDB ;

5. Indexation et gestion de fichiers

Définition générale

En informatique, la notion d'index concerne d'abord les fichiers et les enregistrements (fiches enregistrées) dans les fichiers.

Chaque enregistrement possède une clé primaire.

Un index est un tableau qui associe à chaque fiche du fichier son adresse relative dans le fichier.

Plus précisément, l'index associe la valeur de la clé de chaque fiche à l'adresse relative de la fiche dans le fichier.

L'index d'un fichier F peut être rangé dans un nouveau fichier ou dans le fichier F, à la fin par exemple (c'est le cas le plus courant).

Exemple de fichier avec un index à la fin :

	F7			F4		F1				F3	F7	0	F4	3	F1	5	F3	9
Adresse :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Le fichier contient 4 fiches de tailles variables. Les clés de ces fiches sont : F7, F4, F1, F3.

La clé de chaque fiche est en général placée au début de la fiche.

A noter qu'en principe l'index n'est pas forcément trié.

Caractéristiques générales de l'indexation

Index trié

Un index peut être trié ou pas. En général, on utilise plutôt les index triés.

Fichier trié

Un index peut s'appliquer à un fichier trié ou pas.

Index dense et non dense

La densité d'un index c'est le rapport entre le nombre d'éléments indexés et le nombre total d'éléments (autrement dit, dans le cas d'une BDR, entre le nombre de lignes de la table d'index et le nombre de lignes de la table d'origine).

La densité d'un index varie entre 0 et 1.

Quand la densité vaut 1, tous les éléments du fichier (ou de la table) sont indexés. On parle d'index dense.

Les 8 différents cas possibles

Cas	Tri du fichier	Densité	Tri de l'index	Applications
1	Non trié	Non dense	Non trié	Impossible
2	Non trié	Non dense	Trié	Impossible
3	Non trié	Dense	Non trié	∅ : évolue en 4
4	**Non trié**	**Dense**	**Trié**	**IS3**
5	Trié	Non dense	Non trié	∅ : évolue en 6
6	**Trié**	**Non dense**	**Trié**	**ISAM, VSAM, UFAS**
7	Trié	Dense	Non trié	∅ : évolue en 6
8	Trié	Dense	Trié	∅ : lourd : évolue en 6

ISAM : Indexed Sequential Acces Method. Utilisé par DOS et MVS-IBM.
VSAM : Virtual Sequential Acces Method. C'est un ISAM amélioré qui utilise des arbres B+.
UFAS : inspire de VSAM, c'est une méthode de BULL.
IS3 : Indexed Serie 3. Utilisé sur les AS400 d'IBM. C'est un ISAM dérivé avec arbres B+.
Ce sont des méthodes de gestion de fichier de certains SE.

6. Organisation des fichiers : arbre, arbre B, arbre B+

Problématique

Les techniques ISAM, VSAM, UFAS utilisent des fichiers triés.

Toutes les techniques utilisent des index triés.

Le problème d'une série triée, c'est l'ajout d'un élément dans le fichier : il faut insérer l'élément au bon endroit. Si la gestion est séquentielle : il faudra déplacer tous ceux du dessous, ce qui est très coûteux.

Le problème se pose de la même façon pour les index si on a des index très grand.

La solution consiste à utiliser une structure d'arbre.

Arbre binaire

Définition

Principes généraux

- Un arbre binaire est une organisation composée de nœuds reliés entre eux par des branches.
- Un nœud peut être parent et/ou enfant.
- Les branches sont orientées : elles vont du nœud parent au nœud enfant.
- Un nœud a au maximum un parent.
- L'unique nœud de l'arbre qui n'a pas de parent est appelé « racine ».
- Les nœuds de l'arbre qui n'ont pas d'enfants sont appelés « feuille ».

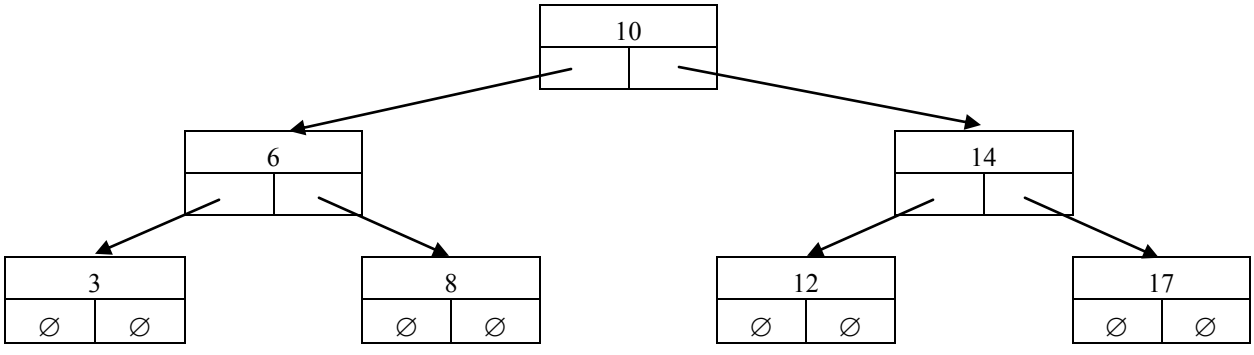
Principes spécifiques

- Un nœud a au maximum deux enfants.
- Chaque nœud porte l'information d'un enregistrement et un seul (une fiche ou un tuple par exemple).

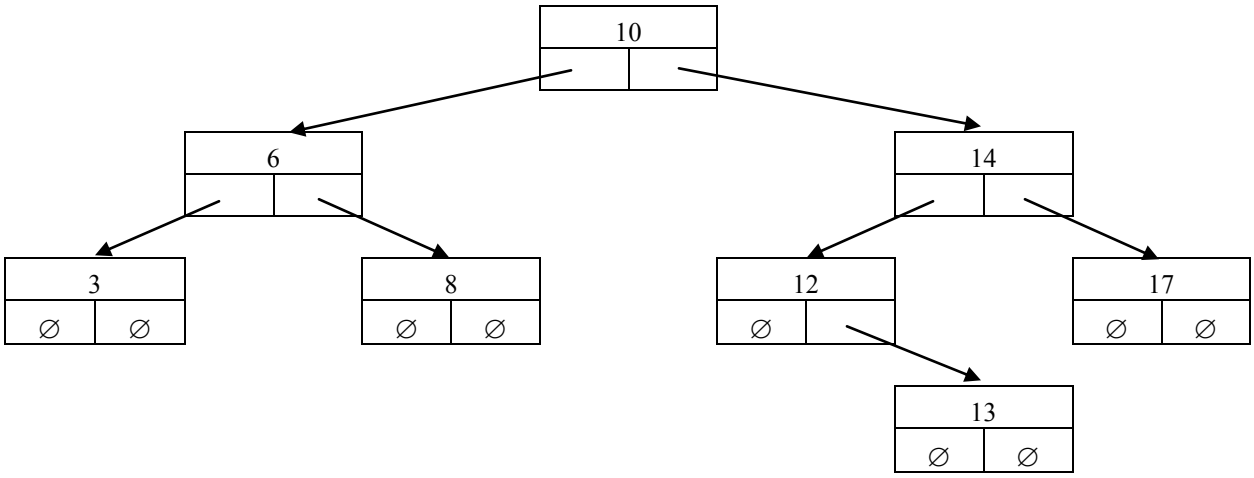
Usage

Un arbre binaire permet de faire des recherches dichotomiques et d'insérer assez facilement de nouvelles données.

Exemple d'arbre binaire



Si on veut ajouter 13 dans cet arbre, on le place sous le 12, à droite :

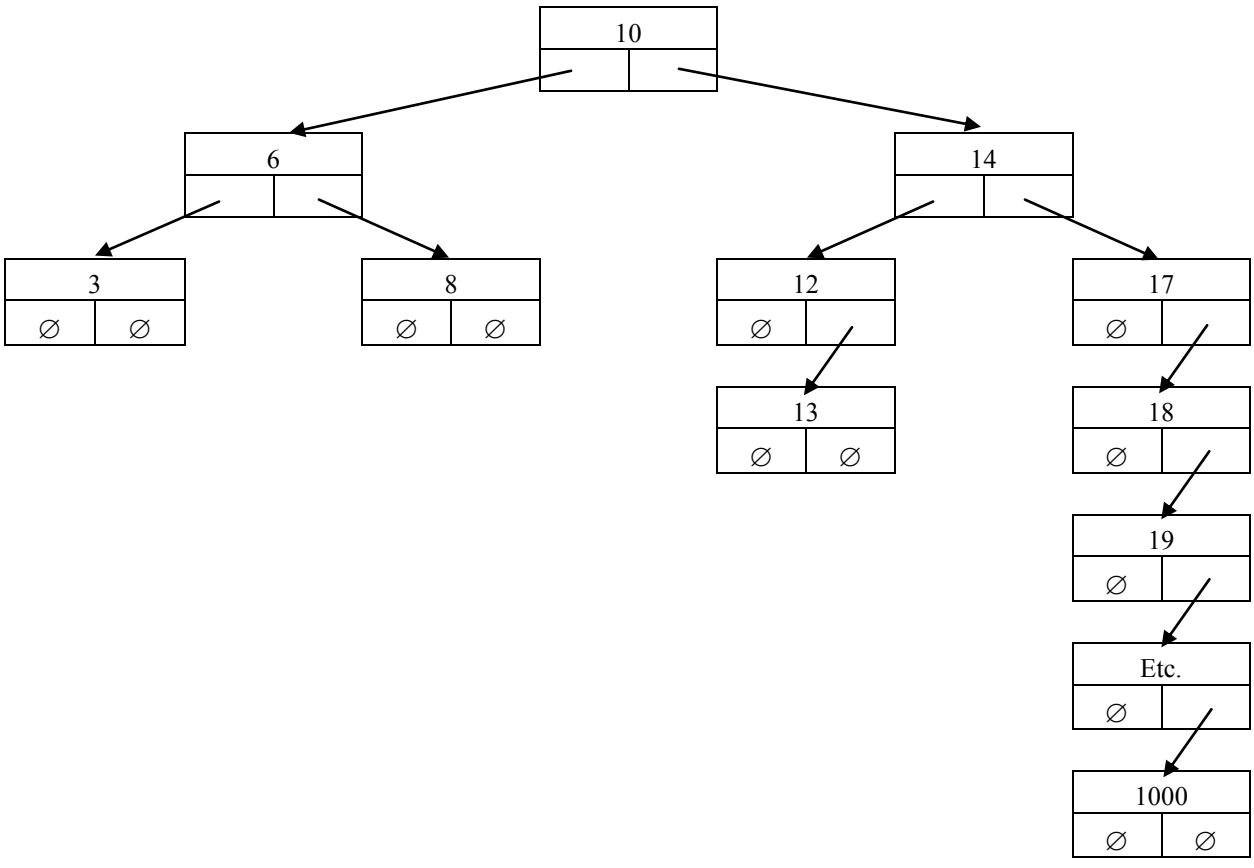


Contenu de chaque nœud

Un nœud contient la valeur et l'adresse de ses deux fils.

Déséquilibre de l'arbre binaire

Dans notre exemple, si on ajoute successivement 18, 19, 20, 21, etc. jusqu'à 1000, on va obtenir un arbre complètement déséquilibré avec une grande série mono-branche.



Autrement dit, l'arbre binaire se transforme en liste chaînée. De ce fait, la recherche de la valeur 1000 dans un tel arbre se fera en presque 1000 tests.

Solution du problème du déséquilibrage

On peut soit ajouter en maintenant l'équilibre : l'algorithme est complexe et surtout potentiellement coûteux.

On peut aussi équilibrer occasionnellement indépendamment des ajouts, par exemple la nuit.

La solution la plus adaptée dépend des usages de la BD.

Arbre B (arbre balancé, B-tree)

Définition

Principes généraux

- Un arbre B (ou arbre balancé ou B tree) est une organisation composée de nœuds reliés entre eux par des branches.
- Un nœud peut être parent et/ou enfant.
- Les branches sont orientées : elles vont du nœud parent au nœud enfant.
- Un nœud a au maximum un parent.
- L'unique nœud de l'arbre qui n'a pas de parent est appelé « racine ».
- Les nœuds de l'arbre qui n'ont pas d'enfants sont appelés « feuille ».

Principes spécifiques

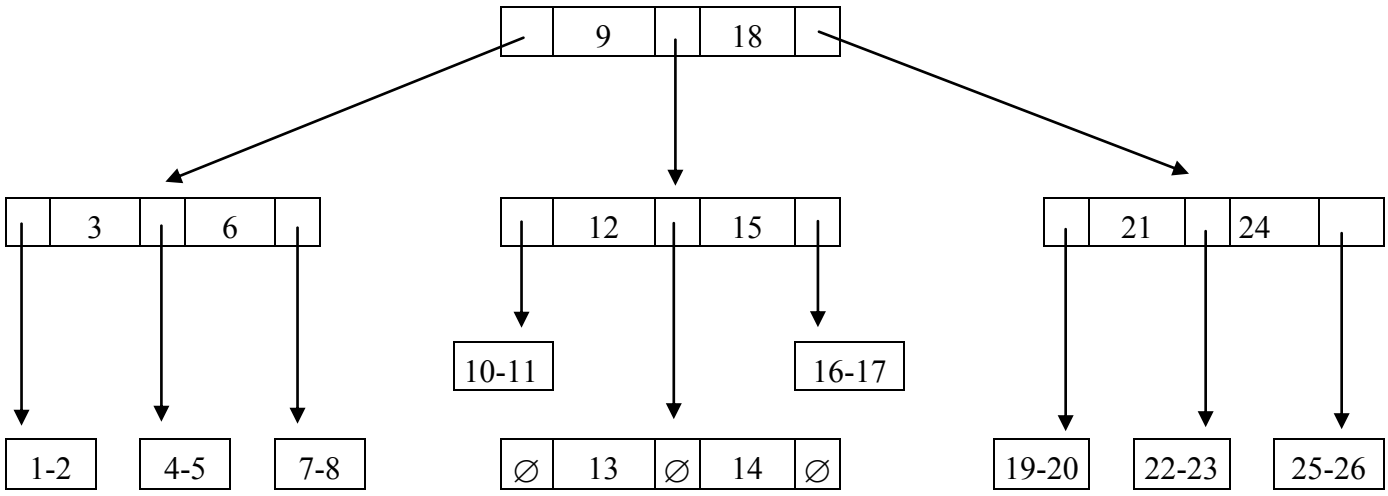
- Un arbre B a un numéro d'ordre : N.
- Les nœuds ont au maximum 2 * N + 1 enfants.
- Les nœuds non-feuilles et non racines ont au minimum N + 1 enfants.
- Chaque nœud non-feuille porte NF-1 enregistrements, avec NF = nombre de fils du nœud.
- Chaque nœud feuille porte 2* N enregistrements au maximum (2*N fiches ou 2*N tuples par exemple).

Bilan

Ordre	Nb fils min	Nb fils max	Nb info max/noeud
N	N+1 (sauf racine et feuilles)	2*N + 1 = NFM	Nb Fils –1 = NFM - 1
1	2	3	2
2	3	5	4
3	4	7	6
…			
10	11	21	20

Exemple

Arbre B d'ordre 1 parfaitement équilibré :



Nombre maximum d'éléments dans un arbre B

Le nombre maximum d'éléments dans un arbre B est fonction de l'ordre N de l'arbre et de la hauteur de l'arbre H. On considère qu'un arbre avec seulement un nœud racine est de hauteur 0.

Le nombre d'éléments maximum est donné par la formule :

$$\mathbf{2*N * Somme(i :0 à H) (2*N +1)^i}$$

Dans le cas d'un arbre d'ordre 1 (3 fils max et 2 éléments max par nœud) et de hauteur 2, on obtient :

$$2 * (3^0 + 3^1 + 3^2) = 2 * (1 + 3 + 9) = 26$$

Dans le cas d'un arbre d'ordre 2 (5 fils max et 4 éléments max par nœud) et de hauteur 5, on obtient :

$$4 * (5^0 + 5^1 + 5^2 + 5^3 + 5^4) = 4 * (1 + 5 + 25 + 625 + 3125) = 15\ 624$$

Dans le cas d'un arbre d'ordre 5 (11 fils max et 10 éléments max par nœud) et de hauteur 5, on obtient :

$$10 * (11^0 + 11^1 + 11^2 + 11^3 + 11^4) = 1\ 771\ 560$$

Le problème de l'ajout et de la suppression dans les arbres B

L'ajout et la suppression dans les arbres B posent le même problème que pour les arbres binaires : celui du maintien de l'équilibre de l'arbre.

Choix du numéro d'ordre d'un arbre

Le nœud d'un arbre va correspondre à la taille d'un bloc sur disque (par exemple 512 octets).

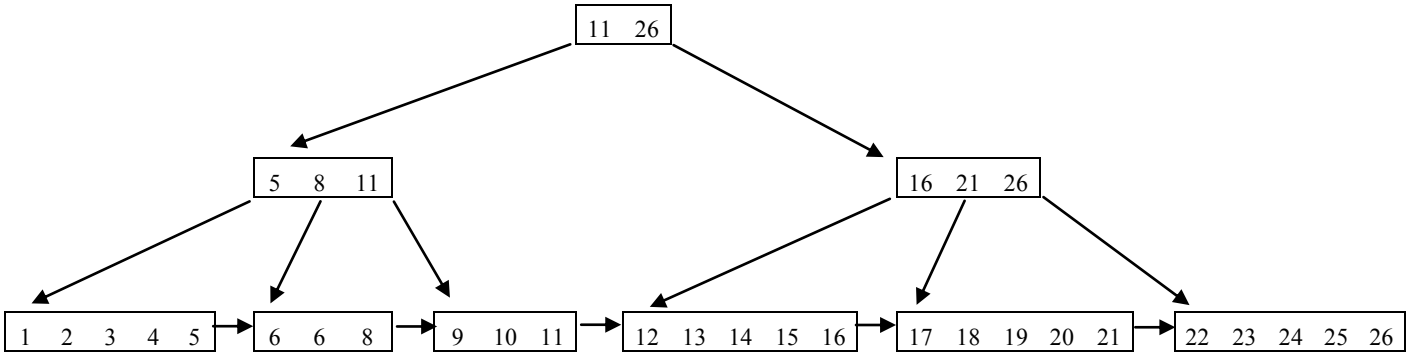
Ainsi, on limitera les accès disques au maximum.

Arbre B+

L'utilisation d'arbres B peut s'avérer coûteuse pour un traitement séquentiel du fait du parcours de l'arbre.

L'arbre B+ permet de pallier à cette difficulté en répétant les informations figurant dans les nœuds parents au niveau des nœuds enfants et en liant les nœuds feuilles entre eux.

De ce fait, on retrouve une structure séquentielle en parcourant directement tous les nœuds feuilles.



En pratique, on utilise les arbres B+ :

- Soit pour implémenter seulement les index. Cela correspond à la méthode IS3 (index dense trié, fichier non trié).
- Soit pour implémenter les index et les fichiers. Cela correspond aux méthodes VSAM et UFAS (index non dense trié, fichier trié).

7. Organisation des index : index hiérarchisés

Problématique

Un index est un tableau trié.

A chaque ajout ou suppression d'un élément dans la table d'origine, l'index doit être mis à jour.

Le problème revient alors à celui de l'ajout et de la suppression dans un tableau trié : à chaque opération, il faut déplacer les éléments du dessous. Si l'index est très grand, l'opération sera coûteuse.

On va donc choisir d'organiser l'index non plus dans un tableau mais dans un arbre.

Notion d'index hiérarchisé

Un index organisé dans un arbre est appelé index hiérarchisé ou index d'index.

HACHAGE

1. Présentation

Les méthodes d'accès par hachage sont basées sur l'utilisation d'une **fonction de calcul** (la fonction de hachage) **qui, appliquée à la clé, détermine l'adresse relative d'un paquet** (bucket en anglais) ou se trouve l'article. (c. Gardarin , Bases de données, p. 73).

On distingue hachage statique et hachage dynamique.

Le hachage statique concerne les fichiers de taille fixe.

Le hachage dynamique concerne les fichiers de taille variable.

2. Le hachage statique

Principe

Du fait de la taille constante du fichier, la méthode est simple.

Le fichier est divisé en P paquets de taille T fixe. L'adresse relative de chaque paquet est donnée par la formule : AR = NB * T, avec NB : n° du bloc en question.

Ensuite, dans le paquet, les articles sont rangés dans leur ordre d'arrivée. En tête de chaque paquet, on trouve l'adresse du premier octet libre du paquet.

Dans le paquet, l'accès est séquentiel ou direct si on ajoute un autre système d'accès dans le paquet.

Fonction de hachage

Il faut calculer le numéro du paquet à partir de la clé d'un article.

La technique la plus répandue est celle du modulo. La clé doit être un entier. La technique consiste alors à prendre pour numéro de paquet le reste de la division de la clé par le nombre de paquets.

Le choix de la fonction de hachage est primordial pour assurer une équi-répartition des articles dans les N paquets. Ce choix doit être guidé par la distribution, rarement uniforme, des clés dans le fichier.

Problème des collisions (ou débordement)

Il y a collision ou débordement quand un paquet est plein et qu'on veut y mettre un nouvel article.

Plusieurs solutions techniques existent.

Le plus souvent, ces techniques utilisent un paquet de débordement, éventuellement avec une deuxième fonction de hachage pour accéder à l'article dans ce paquet.

Dans tous les cas, la gestion du débordement dégrade les performances et complique les algorithmes.

Conclusion

Avantages

Méthode simple et performante. Une lecture d'article s'effectue en une entrée-sortie.

Inconvénients

La taille du fichier doit être fixée a priori

Le débordement dégrade les performances en augmentant le nombre d'entrée-sortie.

3. Le hachage dynamique

Principe

L'objectif des organisations par hachage dynamique est de minimiser le coût des collisions en substituant aux techniques de débordement un accroissement dynamique de la taille du fichier.

Du fait de la taille variable du fichier, la méthode est plus complexe.

Le principe est d'utiliser une fonction de hachage de la clé qui génère une chaîne de N bits où N est grand (par exemple 32).

Lors de la première implantation du fichier haché, seuls les M premiers bits sont utilisés (avec M petit devant N). pour calculer le numéro du paquet.

Quand un premier paquet déborde, une nouvelle région est allouée au fichier et les articles du paquet plein sont répartis entre le paquet plein et le nouveau paquet.

Les bits M+1, M+2, etc. de la fonction de hachage sont successivement utilisés.

4. Exemple d'optimisation avec du hachage

Créer un attribut « haché » basé sur l'information d'autres attributs : c'est plus efficace q'un index sur de nombreuses colonnes.

MOTEURS DE STOCKAGE ET TYPES DE TABLE

1 : Les différents moteurs de stockage et types de tables

Présentation

http://dev.mysql.com/doc/refman/5.0/fr/storage-engines.html

MySQL supporte plusieurs moteurs de stockage, qui gère différents types de tables.

Le moteur de tables d'origine était ISAM. Ce moteur a été remplacé par le moteur MyISAM.

- Le moteur **MyISAM** est une version améliorée de ISAM.
- Le moteur **MEMORY** (anciennement appelé HEAP) propose des tables stockées en mémoire.
- Le moteur **MERGE** permet le regroupement de tables MyISAM identiques sous la forme d'une seule table.

Ces trois moteurs sont non transactionnels.

- Les moteurs **InnoDB** et **BerkeleyDB** gèrent des tables transactionnelles.

NDBCluster est le moteur de stockage du cluster MySQL qui implémente des tables réparties sur plusieurs serveurs.

Le moteur MyISAM était le moteur par défaut jusqu'au rachat par ORACLE.

Depuis la version 5.5, le moteur par défaut est InnoDB.

Avantages et inconvénients

Avantages des tables transactionnelles

- Plus sûr. Même si MySQL crashe ou que vous avez un problème matériel, vous pouvez récupérer vos données, soit par un recouvrement automatique, soit à partir d'une sauvegarde combinée avec le log des transactions.
- Vous pouvez combiner plusieurs commandes et les accepter toutes d'un seul coup avec la commande COMMIT.
- Vous pouvez utiliser ROLLBACK pour ignorer vos modifications (si vous n'êtes pas en mode auto-commit).
- Si une mise à jour échoue, tout vos changements seront annulés.
- Gère mieux les accès concurrents si la table reçoit simultanément plusieurs lectures.

Avantages des tables non-transactionnelles

- Plus rapides
- Utilisent moins d'espace disque
- Utilisent moins de mémoire pour exécuter les mises à jour.

Organisation des données sur disque

Chaque table d'une BD est stockée sur disque dans trois fichiers.

Les fichiers portent le nom de la table, et ont une extension qui spécifie le type de fichier.

Le fichier .frm stocke la définition de la table.

Le fichier des données possède l'extension .MYD (MYData).

Le fichier d'index possède l'extension .MYI (MYIndex),

Pour spécifier explicitement que vous souhaitez une table MyISAM, indiquez le avec l'option ENGINE ou TYPE lors de la création de la table.

MySQL crée toujours un fichier .frm pour stocker le type de la table et les informations de définition. Les données et les index de la table peuvent être stockés ailleurs, en fonction du type de tables.

2 : Le moteur MyISAM

Caractéristiques principales

Le moteur MyISAM est un moteur non transactionnel et qui ne prend pas en compte les contraintes de clé étrangère (FOREIGN KEY).

Type par défaut

En général, MyISAM est le type de table par défaut, à moins d'avoir été spécifié autrement.

Quelques caractéristiques particulières des tables MyISAM

- Le nombre maximum d'index par table est de 64.
- Le nombre maximum de colonnes par index est 16.
- La taille maximum d'une clé est de 1000 octets.
- Les colonnes BLOB et TEXT peuvent être indexées (Les types BLOB correspondent à est un objet binaire de grande taille qui peut contenir une quantité variable de données. Les quatre types BLOB (TINYBLOB, BLOB, MEDIUMBLOB, et LONGBLOB) ne différent que par la taille maximale de données qu'ils peuvent stocker. Les 4 types TEXT (TINYTEXT, TEXT, MEDIUMTEXT, et LONGTEXT) sont équivalents aux quatre types BLOB, mais ils sont insensibles à la casse en cas de comparaison et donc de tri).
- Les valeurs NULL sont autorisées dans une colonne indexée.
- La gestion interne des colonnes AUTO_INCREMENT. MyISAM va automatiquement modifier cette valeur lors d'une insertion ou d'une modification.
- La valeur courante d'AUTO_INCREMENT peut être modifiée avec un ALTER TABLE ou myisamchk.
- Vous pouvez insérer de nouvelles lignes dans une table qui n'a aucun bloc vide dans le fichier de données, en même temps que d'autres threads lisent le fichier de données (insertion simultanée). Un bloc vide peut provenir d'une modification de ligne à format

dynamique (les données sont maintenant plus petites). Lorsque tous les blocs vide sont à nouveau utilisés, les insertions suivantes peuvent être simultanées.

- Chaque colonne de caractères peut avoir un jeu de caractères distinct.
- Support du vrai type VARCHAR jusqu'à 64 Ko (sous peu).

Format de stockage des tables MyISAM

MyISAM supporte 3 formats de stockage différents : FIXED (statique), DYNAMIC et COMPRESSED.

Les formats FIXED et DYNAMIC sont choisis automatiquement selon le type de la colonne utilisé.

Le format COMPRESSED est créé avec l'outil myisampack.

Quand vous créez une table avec CREATE ou en modifiez la structure avec ALTER vous pouvez, pour les tables n'ayant pas de champ BLOB forcer le type de table en DYNAMIC ou FIXED avec l'option de table ROW_FORMAT.

Caractéristiques des tables statiques

Le format statique est le format par défaut.

Il est utilisé lorsque la table ne contient pas de colonnes de type VARCHAR, BLOB, ou TEXT.

➢ ***Avantages***

- Très rapide.
- Facile à mettre en cache.
- Facile à reconstruire après un crash, car les enregistrements sont localisés dans des positions fixées.
- N'a pas à être réorganisé (avec myisamchk) sauf si un grand nombre de lignes est effacé et que vous voulez retourner l'espace libéré au système d'exploitation.

➢ ***Inconvénients***

- Requière usuellement plus d'espace disque que les tables dynamiques : toutes les colonnes CHAR, NUMERIC, et DECIMAL sont complétées par des espaces jusqu'à atteindre la longueur totale de la colonne.

Caractéristiques des tables dynamiques

Le format dynamique est utilisé si une table MyISAM contient des colonnes de taille variable : VARCHAR, BLOB, ou TEXT, ou si la table a été créée avec l'option ROW_FORMAT=DYNAMIC.

Si un enregistrement devient plus grand, il est divisé en autant de parties que nécessaire, ce qui provoque une fragmentation de l'enregistrement.

➢ ***Avantages***

- Chaque enregistrement n'utilise que la quantité d'espace requise.

➢ ***Inconvénients***

- La fragmentation peut dégrader les performances des requêtes. Il faut exécuter un OPTIMIZE TABLE ou myisqmchk –r de temps en temps pour améliorer les performances.

- La fragmentation peut dégrader les performances de la reconstruction après crash.

3 : Le moteur MERGE

http://dev.mysql.com/doc/refman/5.0/fr/merge-storage-engine.html

Une table MERGE est une collection de tables MyISAM identiques qui peuvent être utilisaées à la manière d'une table unique.

On peut l'apparenter à la notion de « vue ».

L'exemple suivant vous montre comme utiliser les tables MERGE :

```
mysql> CREATE TABLE t1 (
    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    message CHAR(20));
mysql> CREATE TABLE t2 (
    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    message CHAR(20));
mysql> INSERT INTO t1 (message) VALUES('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
    a INT NOT NULL AUTO_INCREMENT,
    message CHAR(20), INDEX(a))
    TYPE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

Une table MERGE ne peut pas contenir de contrainte de type UNIQUE sur toute la table. Lorsque vous faites une insertion, les données vont dans la première ou la dernière table (suivant la méthode d'insertion INSERT_METHOD=xxx) et cette table MyISAM s'assure que les données sont uniques, mais rien n'est fait pour vérifier l'unicité auprès des autres tables MyISAM tables.

4 : Le moteur MEMORY

http://dev.mysql.com/doc/refman/5.0/fr/memory-storage-engine.html

Le moteur de stockage MEMORY (anciennement HEAP) crée des tables dont le contenu est stocké en mémoire.

Les tables MEMORY sont très rapides et très utiles pour créer des tables temporaires.

Toutefois, lorsque le serveur ferme, toutes les données stockées dans les tables MEMORY sont perdues.

Par contre, les tables continuent d'exister. Leurs définitions sont stockées dans les fichiers .frm sur le disque.

5 : Les tables compressées : myisampack

http://dev.mysql.com/doc/refman/5.0/fr/myisampack.html

On peut compresser les tables MyISAM avec l'utilitaire myisampack pour réduire leur taille sur le disque.

Généralement, myisampack compresse le fichier avec un gain de 40 à 70 %.

Les tables compressées peuvent être décompressées avec myisamchk.

Les tables compressées sont en lecture seule.

6 : Le moteur ARCHIVE

http://dev.mysql.com/doc/refman/5.0/fr/archive-storage-engine.html

Le moteur de table ARCHIVE est utilisé pour stocker de grande quantité de données, sans index, et de manière très économique.

Le moteur ARCHIVE ne supporte que les commandes INSERT et SELECT : aucun effacement, remplacement ou modification.

Les enregistrements sont compressé au moment de leur insertion. Vous pouvez utiliser la commande OPTIMIZE TABLE pour analyser la table, et compresser encore plus.

7 : Le moteur InnoDB

http://dev.mysql.com/doc/refman/5.0/fr/innodb.html

Caractéristiques principales

Moteur transactionnel : compatibilité ACID complète.

Prise en compte des contraintes de clé étrangère (FOREIGN KEY).

Performances maximales dors du traitement de grands volumes de données.

Aucune limite de taille de table, même sur des systèmes d'exploitation dont la taille de fichier est limitée à 2Go.

8 : Programmation des tables : rappels

Gestion des bases de données

```
mysql>CREATE database NomBD;
mysql>USE NomBD;
mysql>DROP database if exists NomBD;
```

Création des tables

Création des tables

http://dev.mysql.com/doc/refman/5.0/fr/create-table.html

- ***Syntaxe MySQL***

En première approche, la commande a la syntaxe suivante :

```
CREATE table NomTable  (
        attribut_1  type [contrainte d'intégrité],
        attribut_2  type [contrainte d'intégrité],
        … ,
        attribut_n  type [contrainte d'intégrité] ,
        [contrainte d'intégrité]
) ENGINE moteur;
```

Les contraintes sont facultatives.

L'ordre dans la liste est au choix.

- ***Exemple***

```
CREATE TABLE DEPT (
    ND          integer primary key auto_increment,
    NOM         varchar(14),
    VILLE       varchar(13)
);

CREATE TABLE EMP (
    NE          integer primary key auto_increment,
    NOM         varchar(10) not NULL,
    JOB         varchar(9),
    DATEMB      date,
    SAL         float(7,2),
    COMM            float(7,2),
    ND          integer not null, foreign key(ND) references DEPT(ND),
    NEchef      integer , foreign key(NEchef) references EMP(NE)
) ENGINE InnoDB;
```

Variante :

```
CREATE TABLE EMP (
    NE          integer auto_increment,
    NOM         varchar(10),
    JOB         enum ('PRESIDENT','MANAGER', 'SALESMAN', 'CLERK',
    'ANALYST'),
    DATEMB      date,
    SAL         float(7,2) check (sal >1000), -- mysql ne gère pas le check !
    COMM        float(7,2) default 100,
    ND          integer not null,
    NEchef      integer,
    primary key(NE)
) ENGINE InnoDB;

CREATE TABLE DEPT (
    ND          integer auto_increment,
    NOM         varchar(14),
    VILLE       varchar(13),
```

```
    primary key(ND)
) ENGINE InnoDB;

ALTER TABLE EMP ADD constraint KEYND foreign key(ND) references
    DEPT(ND);
ALTER TABLE EMP ADD constraint KEYNECHEF foreign key(NEchef)
    references EMP(NE);
```

Dans la variante, on crée les FOREIGN KEY avec un ALTER TABLE après la création des tables. Cette technique évite les problèmes d'ordre dans la création des tables.

Les contraintes d'intégrité

- ***Liste des contraintes d'intégrité***
  - **PRIMARY KEY :** permet de définir les clés primaires. Cette contrainte garantit le fait que la valeur est différente de NULL et qu'elle est unique dans la table.
  - **FOREIGN KEY :** permet de définir les clés étrangères. Cette contrainte fait référence à une clé primaire dans une autre table.
  - **NOT NULL :** impose le fait que la valeur de l'attribut doit être renseignée.
  - **UNIQUE :** impose le fait que chaque tuple de la table doit, pour l'attribut concerné, avoir une valeur différente de celle des autres ou NULL.
  - **DEFAULT :** permet de définir une valeur par défaut.
  - **CHECK :**permet de définir un ensemble de valeurs possible pour l'attribut. Cette contrainte garantit le fait que la valeur de l'attribut appartiendra à cet ensemble. MySQL ne gère pas cette contrainte.
  - **ENUM :**permet de définir un ensemble de valeurs possible pour l'attribut. Ce n'est pas une contrainte à proprement parler.
- ***Conséquences des contraintes d'intégrité***
  - **PRIMARY KEY, NOT NULL, UNIQUE et CHECK :** ces quatre contraintes ont le même type de conséquence : si on cherche à donner une valeur à un attribut qui n'est pas conforme à ce qui est précisé dans la définition de l'attribut (valeur NULL s'il est défini NOT NULL ou PRIMARY KEY, valeur existant déjà s'il est défini UNIQUE ou PRIMARY KEY, valeur n'appartenant pas au domaine spécifié par le CHECK), alors le SGBD renvoie un message d'erreur et ne modifie pas la base de données. Ainsi, un premier niveau de cohérence des données est maintenu.
  - **DEFAULT :** donne une valeur par défaut si il n'y a pas de saisie.

Paramétrage des contraintes d'intégrité référentielle

- ***Suppression d'un tuple (1) dont la clé primaire est référencée par d'autres tuples (2)***

  Trois cas peuvent se présenter :

  - Soit on interdit la destruction du tuple (1). Il faudra commencer par détruire les tuples (2) pour pouvoir supprimer le tuple (1). C'est la situation par défaut.
  - Soit le système supprime le tuple (1) et les tuples (2) pour qu'il n'y ait plus de tuples qui fassent référence au tuple (1).

    Dans ce cas on ajoute : **ON DELETE CASCADE** à la définition de la clé étrangère.

Exemple :

ND integer not null, foreign key(ND) references DEPT(ND) **on delete cascade**

Dans notre exemple, cela signifie que si on supprime un département, on supprimera aussi tous les employés du département… ce qui n'est certainement pas un bon choix de modélisation !

- Soit le système supprime le tuple (1) et met la clé étrangère des tuples (2) à NULL pour qu'il n'y ait plus de tuples qui fassent référence au tuple (1).

  Dans ce cas on ajoute : **ON DELETE SET NULL** à la définition de la clé étrangère.

Exemple :

ND integer not null, foreign key(ND) references DEPT(ND) **on delete set NULL**

Dans notre exemple, cela signifie que si on supprime un département, les employés du département auront désormais la valeur NULL comme numéro de département. C'est possible à condition que le numéro de département de l'employé ne soit pas déclaré NOT NULL.

➢ ***Modification d'une clé primaire d'un tuple (1) référencée par d'autres tuples (2)***

Deux cas peuvent se présenter :

- Soit on interdit la modification du tuple (1). Il faudra commencer par modifier les tuples (2) pour pouvoir modifier le tuple (1). C'est la situation par défaut.
- Soit le système modifie le tuple (1) et les tuples (2) pour qu'ils fassent correctement référence au tuple (1).

Dans ce cas on ajoute : **ON UPDATE CASCADE** à la définition de la clé étrangère.

Exemple :

ND integer not null, foreign key(ND) references DEPT(ND) **on update cascade**

Dans notre exemple, cela signifie que si on modifie la clé primaire d'un département, on modifiera aussi les numéros de départements des employés de ce département.

Nommer les contraintes : CONSTRAINT nomContrainte

On peut nommer les contraintes, ce qui permettra ensuite de désactiver et de réactiver les contraintes en y faisant références par leur nom.

Pour cela, il suffit d'ajouter « **CONSTRAINT nomContrainte** » devant la déclaration de la contrainte.

Modification des tables

http://dev.mysql.com/doc/refman/5.0/fr/alter-table.html

Modification des attributs

- ***Ajouter un ou plusieurs attributs à la table :***

```
ALTER TABLE NomTable  ADD (
        attribut_1  type [contrainte],
        attribut_2  type [contrainte],
        … ,
        attribut_n  type [contrainte]
);
```

- ***Modifier un attribut de la table***

```
ALTER TABLE NomTable  MODIFY
        attribut_1  type [contrainte]
;
```

La modification permet d'annuler les contraintes de type NOT NULL ou auto_increment.

- ***Supprimer un attribut de la table***

```
ALTER TABLE NomTable DROP attribut ;
```

Attention

La modification et la destruction des attributs doivent être manipulées avec prudence : une table peut contenir des milliers de données. Il ne faut pas les supprimer ou modifier une table sans précaution.

Modification des contraintes d'intégrité

- ***Ajouter une contrainte***

```
ALTER TABLE NomTable  ADD CONSTRAINT  [contrainte] ;
```

Exemple

```
ALTER TABLE emp
ADD CONSTRAINT  KEYND foreign key(ND) references DEPT(ND);
```

- ***Suppression d'une contrainte nommée***

```
ALTER TABLE NomTable  DROP type de contrainte nom_de_contrainte;
```

Exemple

```
ALTER TABLE emp  DROP foreign key  KEYND;
```

- ***Récupérer le nom des contraintes : show create table*** **maTable**

Si on n'a pas nommé les contraintes à la création, MySQL les nomme automatiquement. Pour récupérer le nom, il faut utiliser la commande

```
Show create table maTable
```

On obtient alors le nom de la contrainte derrière le mot clé CONSTRAINT.

Suppression de la clé primaire

```
ALTER TABLE NomTable  DROP primary key ;
```

On ne peut supprimer la clé primaire que si ce n'est pas un auto incrément, et uniquement si elle n'est pas référencée par une clé étrangère.

Destruction des tables

```
DROP TABLE NomTable  ;
```

Attention :

La modification et la destruction des tables doit être manipulées avec prudence : une table peut contenir des milliers de données. Il ne faut pas les supprimer ou modifier une table sans précaution.

Récupération du code sous mysql : show create table

La commande **show create table** *nomTable* permet de récuperer le code de création d'une table.

Exemple 1 : mysql> show create table dept;

```
CREATE TABLE `dept` (
  `ND` int(11) NOT NULL auto_increment,
  `NOM` varchar(14) default NULL,
  `VILLE` varchar(13) default NULL,
  PRIMARY KEY  (`ND`)
) ENGINE=InnoDB AUTO_INCREMENT=41 DEFAULT CHARSET=latin1
```

- ***Remarques :***

1. le n° du prochain ID automatique sera 41.
2. InnoDB est un choix de SGBD. C'est celui par défaut.
3. Le code est exploitable directement pour créer une table.

Exemple 2 : mysql> show create table emp;

```
CREATE TABLE `emp` (
  `NE` int(11) NOT NULL auto_increment,
  `NOM` varchar(10) default NULL,
  `JOB` varchar(9) default NULL,
  `DATEMB` date default NULL,
  `SAL` float(7,2) default NULL,
  `COMM` float(7,2) default NULL,
  `ND` int(11) NOT NULL,
  `NEchef` int(11) default NULL,
  PRIMARY KEY  (`NE`),
  KEY `ND` (`ND`),
  KEY `NEchef` (`NEchef`),
  CONSTRAINT `emp_ibfk_1` FOREIGN KEY (`ND`) REFERENCES `dept`
     (`ND`),
```

```
  CONSTRAINT `emp_ibfk_2` FOREIGN KEY (`NEchef`) REFERENCES `emp`
    (`NE`)
) ENGINE=InnoDB AUTO_INCREMENT=7944 DEFAULT CHARSET=latin1
```

- ***Remarques :***

1. Le n° du prochain ID automatique sera 7944.
2. Les clés étrangères sont déclarées en plusieurs étapes : avec KEY et avec CONSTRAINT
3. Le code est exploitable directement pour créer une table.

5.6 : Passage d'un moteur à un autre

MyISAM et InnoDB

On peut modifier le moteur d'une table :

- ***Passage en MyISAM***

```
ALTER TABLE emp  TYPE MyISAM  ;
```

Pour passer en d'InnoDB à MyISAM, il ne doit pas y avoir de clé étrangère. Il faut donc commencer par supprimer les contraintes nommées de clé étrangère.

- ***Passage en InnoDB***

```
ALTER TABLE emp  TYPE InnoDB  ;
```

Le passage de MyISAM a InnoDB est possible directement.

BONNES PRATIQUES

Principes généraux

- Indexer les clés primaires
- Indexer les clés étrangères
- Indexer éventuellement les attributs de restriction.
- Evitez les requêtes IMBRIQUEES
- Préférez les requêtes imbriquées qui renvoient des CONSTANTES aux autres.
- Méfiez-vous des vues
- Evitez les DISTINCT inutiles
- Evitez les ORDER BY by inutiles
- Améliorez la taille du buffer dédié au tri (sort_buffer_size dans MySQL)
- Utilisez l'EXPLAIN pour les requêtes critiques
- Indexer les clés primaires
- Indexer les clés étrangères
- Indexer les attributs de restriction
- Utiliser des types de données le plus compacte possible (SMALLINT à la place de INTEGER)
- Eviter les valeurs NULL
- Utiliser des types de données de taille fixe plutôt que de taille variable
- Evitez le « select * » et projetez uniquement les champs utiles.
- Utilisez UNION ALL à la place de UNION, si possible : UNION ALL ne fait pas de « distinct » alors que UNION en fait un systématiquement.
- Utilisez COUNT(*) plutôt que COUNT(colonne), si possible.
- Attention aux opérateurs d'inégalité ou de manipulation de chaînes de caractères et aux fonctions en général sur les attributs indexés : ils peuvent annuler l'utilisation de l'index.

Principes spécifiques

MySQL

- OPTIMIZE : permet de récupérer des emplacements vides dans le fichier contenant les données d'une table. Cette réorganisation accélérera les requêtes.
- mysql_store_result, mysql_use_result : la première instruction demande que la requête soit entièrement traitée sur le serveur avant de fournir les résultats au client. La deuxième demande que les résultats soient fournis au client en pipelining.

- log_slow_queries et long_query_time : log_slow_queries permet d'activer la journalisation des requêtes lentes : celles dont la durée sera supérieure à long_query_time. Malheureusement, le long_query_time s'exprime en secondes et vaut 1 au minimum.

TP

1. BD buveurs

1. Charger la BD : BuveursMyISAM.txt
2. Répondre à requête : quels sont tous les buveurs ayant bu un Bordeaux de degré supérieur ou égal à 13 degré ?
3. Faire le graphe de la question.
4. Consulter le résultat de la commande : show table status.
5. Pour chaque table :
   - Vérifier le code : show create table
   - Observer les index : show index
6. Faites un « explain » de la requête : essayer de comprendre la circulation
7. Quel(les) indexation(s) peut-on envisager pour optimiser la requête ?
8. Faites un « explain » de la requête après ajout d'indecx: essayer de comprendre la circulation et les différences.
9. Quelles sont toutes les optimisations qu'on pourrait envisager ? Dans la BD en général, pour la requête en particulier.
10. Faites un « explain » de la requête après ajout d'indecx: essayer de comprendre la circulation et les différences.