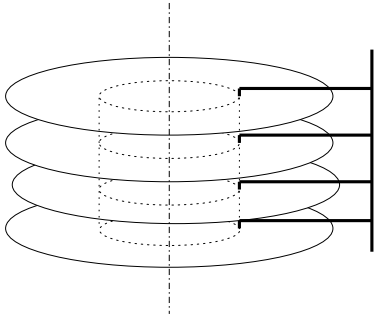
Chapitre VI

L'implémentation du modèle relationnel

Les disques et leurs caractéristiques

Pour comprendre le fonctionnement d'un système de bases de données, et en particulier les aspects de performance, il faut tenir compte du fait que les données sont conservées sur des disques magnétiques.



Organisation d'un disque

- Un disque comporte un ensemble de *surfaces*.
- Sur chaque surface on trouve un ensemble des *pistes* (*tracks*) concentriques.
- Chaque piste est divisée en *secteurs* (*sectors*) de capacité fixée (par exemple 512 octets). Les secteurs sont les plus petits groupes de données adressables et transférables sur un disque.
- Un *cylindre* (*cylinder*) est l'ensemble des pistes se trouvant à une position identique sur les différentes surfaces.

Les secteurs sont parfois aussi appelés *blocs* (*blocks*), mais le terme bloc est aussi utilisé pour des groupes de secteurs traités logiquement comme une seule entité au niveau du système de gestion du disque.

Les performances d'un disque

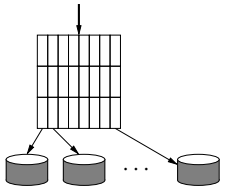
- Pour accéder à un secteur, il faut positionner les têtes de lecture du disque sur le bon cylindre (temps de *recherche* - *seek time*) et
- attendre que le bon secteur passe sous les têtes de lecture (temps de *latence* - *latency*).
- Le temps de recherche plus le temps de latence représente une durée moyenne de l'ordre de 10ms.
- La lecture de secteurs consécutifs est par contre beaucoup plus rapide, de l'ordre de 10 μs par secteur.
- Pour obtenir de plus grosses capacités et de meilleures performances, on utilise des ensembles de disques qui sont gérés comme un seul disque virtuel. C'est ce qu'on appelle un RAID *Redundant Array of Inexpensive Disks*.

La technologie RAID : les niveaux

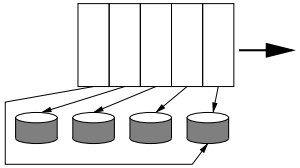
Plusieurs politiques de gestion des RAID ont été définies.
Traditionnellement on parle de *niveaux* RAID.

– Une technique de base qui permet une augmentation des performances est le *striping* (écriture par bandes). Cette technique consiste à distribuer les données entre les disques du RAID en vue de paralléliser les accès. Elle correspond au RAID *niveau 0*. On distingue deux types de "striping".

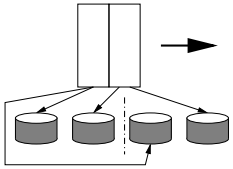
– Le "striping" au niveau du bit (*bit level*). Il correspond à créer des groupes de 8 disques et à répartir les bits de chaque octet entre ces 8 disques.



– Le "striping" au niveau du bloc (*bloc level*). Il correspond à utiliser n disques et à écrire le bloc i sur le disque $(i \bmod n) + 1$.



– Pour augmenter la fiabilité, une technique efficace est de partitionner l'ensemble des disques en 2 groupes et de maintenir sur chacun des deux groupes une copie complète des données. Chaque opération d'écriture est donc réalisée de façon simultanée et symétrique sur chacun des 2 groupes. C'est ce qu'on appelle le *mirroring* (écriture miroir) qui correspond au RAID *niveau 1*.



- Les autres niveaux RAID (2-6) prévoient une redondance par l'utilisation de codes de correction d'erreur.
- L'écriture par bandes (*striping*) peut être combinée avec l'écriture miroir (*mirroring*) ou avec les codes de correction d'erreur.
- Avec une redondance suffisante, on arrive à une excellente fiabilité et on peut même remplacer les disques défaillants sans arrêt du système. C'est ce qu'on appelle un *échange à chaud* (*hot swapping*).

L'implémentation des relations : les fichiers

- Pour permettre l'utilisation aisée des disques, les systèmes d'exploitation fournissent un système de gestion de *fichiers* (*files*).
- Un fichier est un ensemble de données auquel on attribue un nom.
- Les noms des fichiers sont gérés dans un *catalogue* (*directory*) hiérarchique qui permet de retrouver rapidement les fichiers conservés sur un disque.
- Les fichiers de base disponibles dans un système d'exploitation sont vus comme une (longue) séquence d'octets destinée à être lue et écrite séquentiellement.
- Pour implémenter une base de données, il faut des fichiers organisés sous la forme d'un ensemble de tuples ou *enregistrements* (*records*) et des techniques permettant de retrouver rapidement un enregistrement à partir de sa clé (ou à partir d'autres attributs).

Les fichiers ISAM

Les systèmes de bases de données utilisent des fichiers ISAM (Indexed Sequential Access Method) pour conserver les relations.

- Un fichier ISAM est constitué d'une séquence de blocs dans lesquels sont conservés les enregistrements.
- Un parcours séquentiel du fichier est possible, mais on y ajoute des *index* (*indexes*) permettant de retrouver rapidement le bloc dans lequel se trouve un enregistrement.
- Habituellement, on construit un index pour la clé des enregistrements et, suivant les besoins, pour d'autres attributs.

L'implémentation des fichiers ISAM : les B-trees

- L'organisation adoptée pour les index est le *B-tree* qui est une forme d'arbre équilibré optimisé en vue de l'utilisation sur disque.
- Dans un B-tree l'équilibre de l'arbre est maintenu en le réorganisant périodiquement, mais aussi en laissant varier le nombre de successeurs de chaque noeud dans certaines limites.
- Une implémentation optimale sur disque du B-tree utilise des noeuds dont la taille est celle d'un secteur.

Pour un fichier de n enregistrement, rechercher un enregistrement nécessite un temps $O(n)$ sans index, mais seulement $O(\log(n))$ avec un index !

Par contre, plus il y a d'index pour un fichier, plus les opérations de modification seront lentes.

Les tables “hash”

Les tables “hash” sont une deuxième forme d’index qui est aussi fréquemment utilisée, mais plutôt pour des relations temporaires conservées en mémoire. Le principe d’une table “hash” est le suivant.

- Les enregistrements sont répartis entre des bacs regroupés dans un tableau. Un bac peut contenir un ou plusieurs enregistrements (chaînage).
- On prévoit un nombre de bacs qui est supérieur au nombre d’enregistrements prévus, ce qui donne un nombre moyen d’enregistrements par bac inférieur à 1.
- Le numéro de bac dans lequel un enregistrement est placé est calculé à partir de la clé de l’enregistrement par une fonction “hash”. Une fonction “hash” idéale répartirait les enregistrements uniformément entre les bacs.
- Pour avoir accès à un enregistrement, il suffit de calculer son numéro de bac à l’aide de la fonction “hash”.

Les tables “hash” (suite)

- Les fonctions “hash” parfaites n’existent pas, mais il est possible de trouver des fonctions “hash” qui donnent d’excellents résultats en pratique.
- Le temps d’accès à un enregistrement dans une table “hash” comportant n enregistrements est $O(1)$, c’est à dire borné indépendemment de n.

Les index en SQL

- Les index sont spécifiés au moment où une table est créée.
- En général un index est crée par defaut pour la clé primaire.
- Il est possible d’ajouter ou de supprimer un index d’une table : **create index**, **drop index** (cas particuliers de **alter table**).
- il est parfois possible de préciser si un index doit être réalisé à l’aide d’un B-tree ou d’une table “hash”.

L’implémentation des opérations de l’algèbre relationnelle

L'implémentation des opérations : paramètres et critère de performance

Critère de performance : Temps de calcul

Paramètres utilisés (pour chaque relation r) pour estimer le temps de calcul :

n_r **:** nombre de tuples de r

$V(X, r)$ **:** nombre de valeurs distinctes qui apparaissent dans r pour les attributs X
= nombre de tuples distincts dans $\pi_X r$
Le nombre d'occurences moyen de tuples ayant les mêmes valeurs pour les attributs X est donc $\frac{n_r}{V(X,r)}$.

s_r **:** taille des tuples (enregistrements) ;
permet par exemple de déterminer le nombre d'enregistrements par bloc.

Opérations booléennes

$r_1 \cup r_2$ **:**
Implémentation directe : complexité $O(n_{r_1} + n_{r_2})$.
Si l'on désire éliminer les occurences multiples, on peut
– trier le résultat : $O((n_{r_1} + n_{r_2}) \log(n_{r_1} + n_{r_2}))$;
ou
– utiliser un index existant, par exemple pour r_1, et n'ajouter à r_1 que les tuples de r_2 qui ne figurent pas dans r_1 : $O(n_{r_1} + n_{r_2} \log(n_{r_1}))$.

$r_1 - r_2$ **:** On peut
– trier les deux relations et les parcourir ensuite séquentiellement, éliminant de r_1 les tuples de r_2 : $O(n_{r_1} \log(n_{r_1}) + n_{r_2} \log(n_{r_2}))$;
ou
– utiliser un index existant pour r_2 : considérer chaque tuple de r_1, le chercher dans r_2 en utilisant l'index : $O(n_{r_1} \log(n_{r_2}))$.

Sélection et projection

$\sigma_F r_1$ **:**
– Au pire, considérer chaque tuple de r_1 et tester la condition F : $O(n_{r_1})$.
– Une meilleure complexité peut être obtenue s'il y a des index adaptés à la condition de sélection.

$\pi_X r_1$ **:**
– En général : parcours de r_1 avec extraction des composantes nécessaires de chaque tuple : $O(n_{r_1})$.
– Si l'élimination des occurences multiples est nécessaire, alors il faut trier la relation r_1 : $O(n_{r_1} \log(n_{r_1}))$

Produit cartésien et joint

$(r_1 \times r_2)$:

Par définition, le produit cartésien contient $n_{r_1} \times n_{r_2}$ tuples et nécessite un temps de calcul $O(n_{r_1} \times n_{r_2})$.

$(r_1 \bowtie r_2)$:

Stratégies différentes suivant qu'il existe ou non des index pour l'une ou l'autre relation.

Joint sans index

Soit X les attributs communs de R_1 et R_2.
Plusieurs méthodes sont possibles.

Méthode naïve : Examiner toutes les paires de tuples et comparer les valeurs des attributs communs : $O(n_{r_1} \times n_{r_2})$.

Tri et fusion : Trier les deux relations par rapport à leurs attributs communs. Fusionner les résultats.

Complexité : $O(n_{r_1} \log(n_{r_1}) + n_{r_2} \log(n_{r_2}))$, pour autant que $\frac{n_{r_i}}{V(X,r_i)}$ ($i = 1$ ou 2) soit borné pour les attributs communs X.

Joint utilisant une table "hash"

On construit une table "hash" pour r_2 sur les attributs communs X. On parcourt r_1 et pour chaque tuple de r_1 (et sa valeur pour X), on va chercher dans la table "hash" les tuples de r_2 qui ont la même valeur pour X.

Complexité : $O(n_{r_1} + n_{r_2})$, pour autant que $\frac{n_{r_2}}{V(X,r_2)}$ soit borné pour les attributs communs X.

Joint en présence d'un index

Supposons que r_2 possède un index pour les attributs communs X.

On parcourt r_1 et pour chaque tuple de r_1 (et sa valeur pour X), on va chercher dans r_2 les tuples qui ont la même valeur pour X.

Complexité : $O(n_{r_1}(1 + \log(n_{r_2})))$, pour autant que $\frac{n_{r_2}}{V(X,r_2)}$ soit borné pour les attributs communs X.

Optimisation des requêtes

On utilise des équivalences de l'algèbre relationnelle pour minimiser le nombre d'opérations à effectuer pour évaluer une requête.

Soit des relations r de schéma R, s de schéma S et u de schéma U.

– *Associativité et commutativité du joint.*

$$\begin{aligned} r \bowtie s &= s \bowtie r \\ r \bowtie (s \bowtie u) &= (r \bowtie s) \bowtie u \end{aligned}$$

Utilité : Optimiser le calcul de joints multiples.

– *Groupement des conditions de sélection.*

$$\sigma_{F_1}\sigma_{F_2}r = \sigma_{F_1 \wedge F_2}r$$

Utilité :
– Remplacer plusieurs sélections par une seule
– Décomposer une sélection pour exploiter un index
– Décomposer une sélection pour qu'elle puisse en partie être permutée avec une autre opération (voir plus loin)

– *Sélection et projection.*

$$\pi_X\sigma_{A=a}r = \pi_X\sigma_{A=a}\pi_{X\cup A}r$$

Utilité : Permet d'appliquer la sélection à une relation plus petite.

– *Sélection et joint.*
Soit un attribut $A \in R$.

$$\sigma_{A=a}(r \bowtie s) = (\sigma_{A=a}r) \bowtie s$$

Utilité : Réduire la taille des relations avant le calcul d'un joint.

– *Sélection et union ou différence.*

$$\begin{aligned} \sigma_{A=a}(r \cup s) &= (\sigma_{A=a}r) \cup (\sigma_{A=a}s) \\ \sigma_{A=a}(r - s) &= (\sigma_{A=a}r) - (\sigma_{A=a}s) \end{aligned}$$

Utilité : Réduire la taille des relations dès que possible.

– *Projection et joint.*

$$\begin{aligned} &\pi_X(r \bowtie s) \\ &= \pi_X\Big(\pi_{(R\cap(X\cup S))}r \bowtie \pi_{(S\cap(X\cup R))}s\Big) \end{aligned}$$

Utilité : Minimiser la taille des relations avant le calcul du join.

– *Projection et union.*

$$\pi_X(r \cup s) = (\pi_X r) \cup (\pi_X s)$$

Utilité : Minimiser la taille des relations avant le calcul de l'union.

Principes généraux de l'optimisation

- Générer différentes stratégies d'évaluation et en estimer le coût. Choisir la meilleure.
- Règle heuristique : Effectuer les sélections et projections dès que possible.
- Il est utile d'avoir des informations statistiques sur le contenu de la base de données pour bien estimer le coût de différentes stratégies d'évaluation.