Optimisation des performances

Anne-Cécile Caron

Master MIAGE - BDA

1er trimestre 2011-2012

Les tâches de l'administrateur

Sécurité + Performances

- gérer les utilisateurs et leurs droits d'accès
- gérer et optimiser les performances
- gérer les processus.
- gérer le stockage des données
  - organisation des structures logiques et physiques
  - sécurité : sauvegarde, restauration, archivage (cf cours transactions)

mêmes tâches, plus complexes, selon le contexte (SGBD réparti, dataware house, ...)

Comment optimiser les performances ?

- OS et Hardware : 1 ou plusieurs processeurs, configuration des disques, fonctionnement des entrées sorties
- Configuration générale du SGBD :
  - placement des fichiers de données, taille des fichiers
  - transactions : nombre de transactions, type de verrouillage
  - Gestion des log : fréquence des écritures sur le disque

et aussi ...

- Le schéma et les requêtes :
  - la normalisation peut gêner les performances.
  - existence ou non d'index.
  - écriture des modules stockés (par exemple, utilisation d'un curseur explicite plus coûteux qu'une seule requête SQL)
- Application BD : programmer avec un haut niveau d'abstraction peut cacher le coût réel !
  - utilisation de vue : on ne sait pas si sa définition est complexe
  - attention à l'utilisation de procédures "atomiques" : boucle avec une requête réalisée *n* fois plus coûteux qu'une seule requête globale.

Quelques conseils concernant le schéma

- Utiliser intensivement les contraintes d'intégrités (gestion plus efficace par le serveur)
- Utiliser des modules stockés (paquetages, procédures ...)
- Définir correctement les structures logiques et physiques mises en jeu (voir partie Architecture d'Oracle)
- parfois, dénormaliser.
- Optimiser les requêtes
  - créer des index, des clusters (schéma physique, n'apparaît pas au niveau du schéma logique)
  - comparer plusieurs solutions SQL d'une même requête (plan d'exécution)
  - choisir un mode de traitement des requêtes

Indexation

- Un index est une structure de donnée qui permet de retrouver une valeur (ligne) associée à une clé (attention, cette clé (*search key*) n'est pas forcément une clé au sens relationnel (*primary key, candidate key*).
- stockage :
  - Index primaire = l'index et la table sont stockés ensembles. On parle aussi de *clustered index*.
  - Index secondaire = structure supplémentaire, ajoutée à la table.
- Un index peut porter sur un ensemble de colonnes $X_1, \cdots; X_n$: accélère la recherche lorsqu'on connaît $X_1, \ldots, X_i (i \leq n)$.

```
create index idx_nom on employe(nom, prenom);
```

- Un index peut être basé sur une fonction

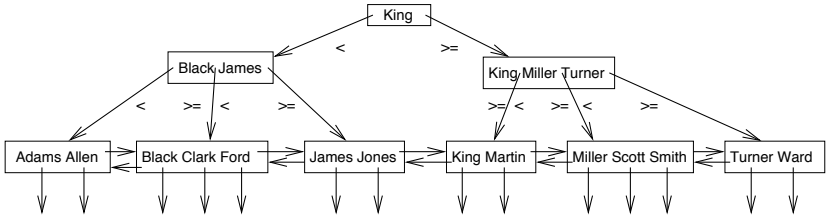
```
create index idx_formule on employe(sal*(1+pct/100));
select * from employe where sal*(1+pct/100) > 1000 ;
```

- Un index peut être unique (la clé de l'index porte sur un ensemble de colonnes unique, souvent clé primaire) ou non unique.

B-arbre

- Toutes les feuilles sont à la même profondeur
- Toutes les clés sont aux feuilles. *ces blocs sont en moyenne au 3/4 pleins*
- Les B-arbres sont équilibrés
- Ces index permettent d'accélérer la recherche
  - Quand on connaît la valeur de la clé de l'index,
  - Quand on fixe un intervalle sur la valeur de la clé de l'index
- Maintenir l'index lors de mise à jour (complexité acceptable).
- Quelques variantes : index unique ou pas, index primaire ou secondaire, clé composée ...

B-arbre, index secondaire, unique



- index secondaire unique : les feuilles contiennent des couples (clé, pointeur vers la donnée)
- index secondaire non unique : il faut gérer des doublons pour une même clé. Sous Oracle, on considère alors que la clé de l'index est composée avec le `rowid` pour avoir un index unique.
- Gestion des clés composées : selon le SGBD, soit on fabrique 1 clé en concaténant les différentes colonnes avec un séparateur, soit on utilise des B-arbres "imbriqués"

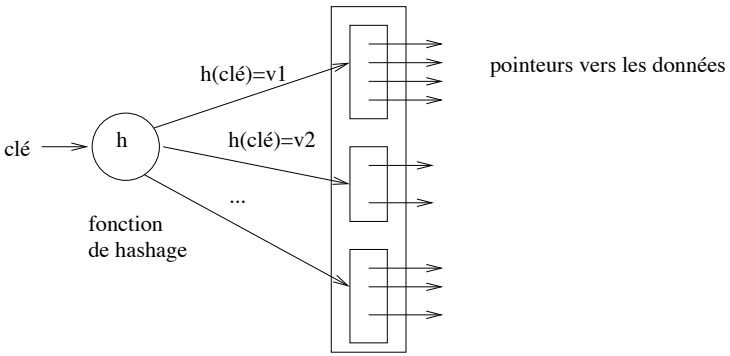
B-arbre comme index primaire

- Parfois, la table est stockée avec l'index : table *index-organized*
- Les feuilles de l'index contiennent alors les données, pas seulement l'adresse d'une ligne stockée ailleurs.
- Dans ce cas, il s'agit d'un index unique, selon la clé primaire.
- Accès plus direct selon la clé primaire, et moins de place en mémoire.
- On peut définir des index secondaires sur une table *index-organized*

Table de hachage

- Une ligne est accessible en appliquant une fonction de hachage sur la clé de la table
- La position des éléments dans une table de hashage est pseudo aléatoire, cette structure de données n'est donc pas adaptée pour accéder à des données triées, contrairement au B-arbre.
- Lorsque deux clés ont la même valeur de hachage, ces clés ne peuvent être stockées à la même position, on doit alors employer une stratégie de résolution des *collisions*.

Table de hachage



sous Oracle

- A chaque définition d'une clé primaire ou d'une contrainte d'unicité, un index est créé. Il en est de même dans les autres SGBD.
- Par défaut, les index sont des B-arbres, index secondaires.
- Il est possible de définir un index primaire : de stocker la table avec son index B-arbre (on parle de table *index-organized*) ou de stocker la table sous forme de table de hachage (on parle de *hash-cluster*)

Cluster

- Groupe de tables qui partagent les mêmes blocs de données parce qu'elles partagent des colonnes (appelées clé du cluster) et sont souvent utilisées ensemble.
- Sans cluster, ces tables seraient stockées dans des blocs différents (des segments de données différents).
- Revient à stocker une jointure (donc optimise la jointure !)
- Economie d'espace
- MAIS insert moins performant.
- Il faut créer un index sur la clé du cluster

Cluster

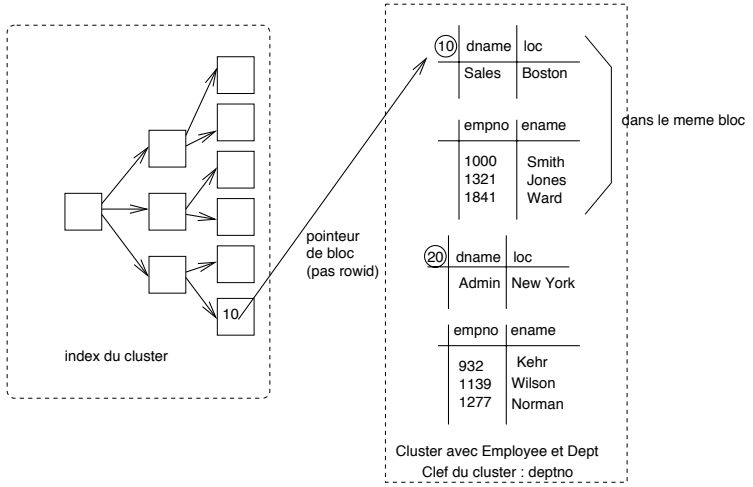


Table de hachage

- Hash Cluster : utilisation d'une fonction de hachage pour localiser les lignes du cluster
- Plus rapide que l'index habituel du cluster, où il est nécessaire de lire plusieurs blocs (sur le chemin entre la racine et la feuille de l'index + accès au bloc du cluster)
- Un tel cluster peut aussi être utilisé pour stocker une seule table : on bénéficie de l'accès par fonction de hachage (table de hachage comme index primaire).

Index Bitmap

- Intéressant quand on utilise comme clé une colonne qui peut prendre peu de valeurs différentes
  situation familiale $\in$ {marié, divorcé, veuf, pacsé, célibataire} ,
  sexe $\in$ {M,F}.
- tableau avec autant de colonnes que de valeurs possibles de la clé, et autant de lignes que dans la table à indexer.
- Chaque case (x,y) contient 1 bit qui indique si la ligne x a pour valeur de clé y, la ligne ne comporte que des 0 si la clé vaut null.

Rowid	M	F
213	1	0
234	0	0
423	1	0
765	0	1
...	...	...