Chapitre 11 : Optimisation
INF3080 BASES DE DONNÉES (SGBD)

Guy Francoeur

3 septembre 2019

UQÀM | Département d'informatique

Matériel et droits

- Les droits de lecture sont accordés aux étudiants inscrits au cours INF3080-030 A2019 uniquement;
- Aucun droit pédagogique ou reproduction n'est accordé sans autorisation;

Table des matières

Table des matières

Retour sur les devoirs

- Questions, précisions, sur les curseurs et les boucles

Table des matières

optimisation

Dans le domaine des bases de données, l'optimisation peut prendre plusieurs aspects. Parmi ceux-ci nous parlerons de coût pour retrouver les records et coût en (KBytes) nécessaire pour le stockage. Le premier est un coût asymptotique c'est-à-dire ... Deuxième aspect est l'espace que nécessite un objet (index) qui aide l'optimisateur (l'optimisation). Nous allons voir des exemples.

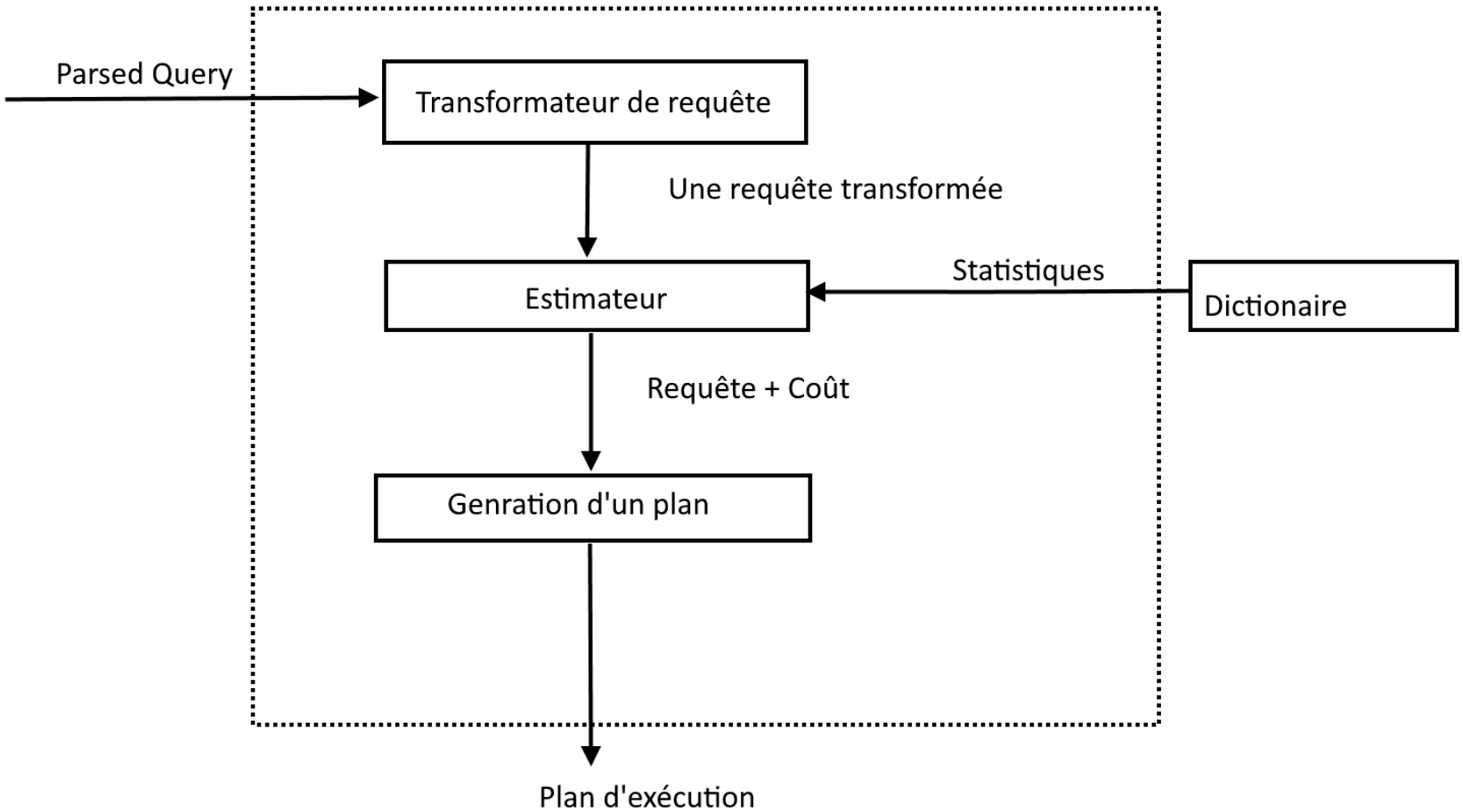
- Quel est le moyen le plus simple pour optimiser une requête ?
- Quels sont selon vous les autres moyens pour optimiser le coût de requêtes ?
- Est-ce toujours complexe ?

l'optimisateur

L'optimisateur de requête doit déterminer la meilleure façon d'exécuter une requête, ceci, en fonction des informations qu'il détient au moment de l'exécution.
Voici les facteurs qui influencent la durée d'exécution d'une requête :

- Full table scans
- Index scans
- Nested loops
- Hash joins

Optimizer


Parsed Query
Transformateur de requête
Une requête transformée
Statistiques
Estimateur
Dictionaire
Requête + Coût
Genration d'un plan
Plan d'exécution

indexes

- index B-TREE
- index BITMAP
- hachache statique / dynamique

Essentiellement, dans Oracle Database, nous avons deux types d'indexage supporter par le SGBDR. Dans les bases de données modernes, il existe aussi des fonctions de hachage qui peuvent dans certains cas améliorer les performances. L'augmentation des performances n'est pas garantie. La complexité asymptotique d'une fonction de hachage n'est pas toujours ajoutée au coût de l'optimisateur de requête.

bitmap index

PARTS table

partno	color	size	weight
1	GREEN	MED	98.1
2	RED	MED	1241
3	RED	SMALL	100.1
4	BLUE	LARGE	54.9
5	RED	MED	124.1
6	GREEN	SMALL	60.1
...	...	...	...

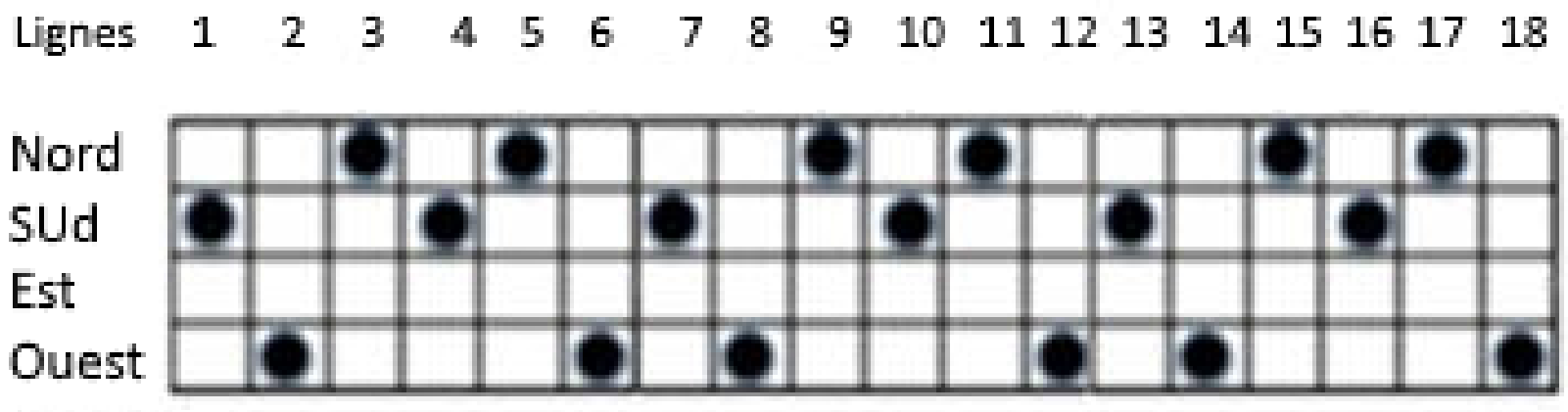
Bitmapped Index on 'color'

color =	'BLUE'	0	0	0	1	0	0	...
color =	'RED'	0	1	1	0	1	0	...
color =	'GREEN'	1	0	0	0	0	1	...
	Part number	1	2	3	4	5	6	

Un exemple de bitmap index sur la couleur pour les 6 premières lignes de la table.

bitmap index

Le point indique la présence de la valeur dans le tuple numéroté.



Le diagramme représente les valeurs distinctes (indexées) à gauche et pour quelles lignes elles sont présentes. 1 veut dire présent et 0 veut dire faux (absent).

bitmap index

Les bitmaps index sont :

- favorables (facile) à la compression;
- utilisables en groupe pour une requête donnée;*
- surtout utilisés sur des clés étrangères;
- inutiles sur des clés uniques;
- favorable sur des colonnes qui changent rarement.

*Une table peut avoir plusieurs bitmaps index et ils seront tous utilisé pour optimiser la découverte des lignes candidates. Par la suite, il y aura une fusion des listes (de lignes) afin de projeter uniquement les lignes communes.

```
SQL> CREATE BITMAP INDEX nom_index on table(colonne);
```

Arbre B

Il existe des variantes de l'arbre B (*B-Tree*). Les variantes existent pour combler des besoins spécifiques. Vous pouvez considérer que c'est une optimisation, une amélioration d'un concept qui est déjà très bon.

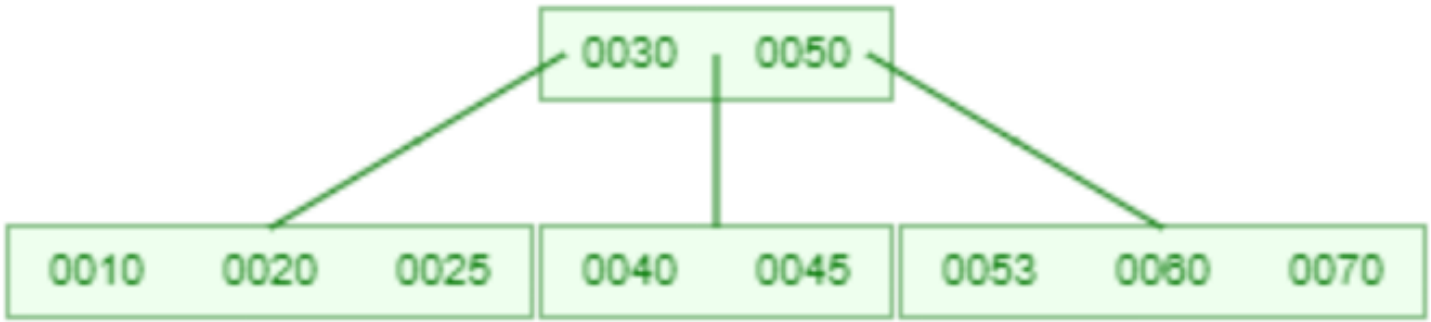
- B+tree. (multiples pointeurs)
- B*Tree ou arbre AVL. (balancé)

Les index dans Oracle Database, lorsque vous ajoutez l'instruction PRIMARY KEY, seront créés à l'aide de l'arbre B (ou une variante). Nous pouvons aussi affirmer que la majorité des bases de données modernes utilisent des arbres B pour indexer les données.

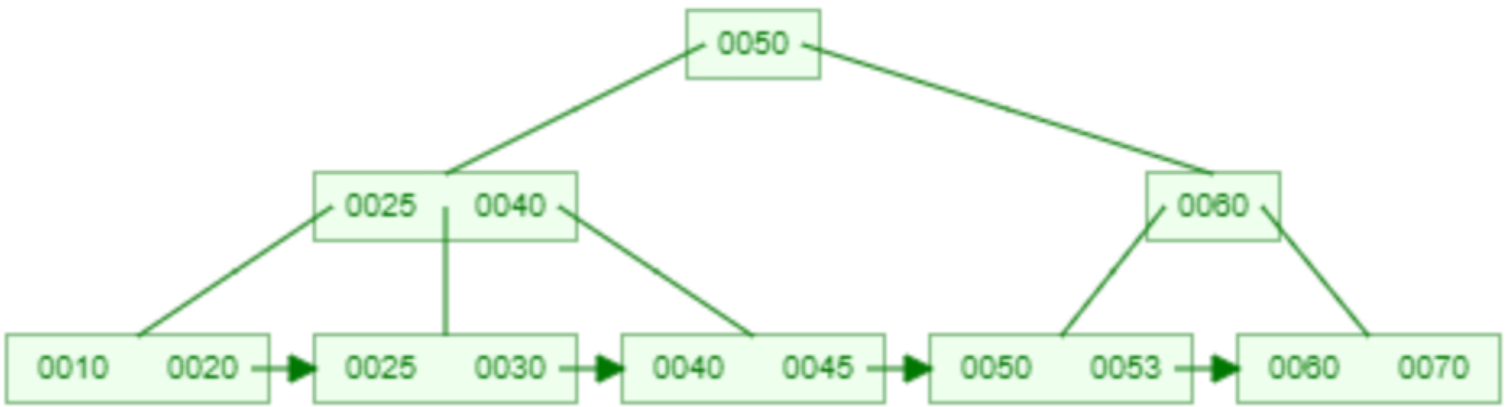
Représentation visuelle

Insertion : $\{40, 20, 60, 30, 25, 10, 70, 50, 53, 45\}$

Consulter BTree



Consulter B+Tree



Arbre B

Les index *B-Tree* sont :

- utiles pour indexer les clés primaires;
- utiles pour recherche rapidement, normalement O(log n);
- non utilisable conjointement;
- non utilisable quand il y a des valeurs nulles dans la colonne;

```
SQL> CREATE INDEX nom_index on table(colonne);
```

Explain Plan

Afin de savoir si les requêtes lancées sont optimales ou efficientes, il est possible de consulter le plan généré de celles-ci.

```
-- nous sommes dans sql*plus
SQL> EXPLAIN PLAN FOR
SELECT count(*) FROM machine where pMachine between 43 AND 100;

SQL> SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

Explain Plan - automatique

Il est possible de ne pas utiliser EXPLAIN PLAN FOR à chaque fois.

```
-- nous sommes dans sql*plus
SQL> set autotrace on explain
SQL> SELECT count(*) FROM Site where pSite BETWEEN 1 AND 123;
```

```
-- nous sommes dans sql*plus
SQL> CREATE BITMAP INDEX idx_modele on Machine(pModele);
SQL> SELECT count(*) FROM Machine WHERE pModele < 123;
```

```
-- nous sommes dans sql*plus
SQL> CREATE BITMAP INDEX idx_machine_revision
on Machine(pRevision);
SQL> SELECT count(*) FROM Machine WHERE pRevision IS NOT NULL
AND pModele IN (1,3);
```

Hachage - définition

- Une fonction de Hachage produit un résultat que nous appelons le hash. Généralement un entier (nombre). Il s'agit d'une transformation unidirectionnelle d'une valeur "chaîne" vers un entier.
- Il peut être utile ou nécessaire d'utiliser une fonction de hachage sur un ensemble de données pour simplifier la comparaison. Exemple, nous voulons comparer deux chaînes de caractères de longueur indéfinie. On veut trouver "Montréal" dans la table on va donc hash toutes les valeurs de cVille vers un entier. Par la suite il sera possible de comparer le standard_hash('Montréal') avec le standard_hash(de toutes les valeurs possibles). La comparaison sera entière.

Hachage - suite

- strcmp("Montréal","Montréal-Ouest") sera potentiellement plus coûteuse qu'un hash statique, si cette comparaison est effectuée plusieurs fois pour effectuer une jointure par exemple. On garde la valeur (hash) pour ne pas le recalculer.
- Un problème avec les fonctions de hachage est qu'elles peuvent retourner deux fois le même résultat pour deux valeurs différentes. On parle ici de collisions. Possible avec des fonctions de hachage de moindre qualité.

```
-- nous sommes dans sql*plus
SQL> SELECT ora_hash('Montreal') FROM dual;

SQL> SELECT standard_hash('New York', SHA512) FROM dual;  --v12c
```

Java + Hachage sha256 (patch pour v11)

```
SQL> --nous sommes dans sql*plus
CREATE OR REPLACE AND RESOLVE JAVA SOURCE NAMED "calcsha"
   AS import java.security.MessageDigest;
public class calcsha256 {
 static public String fsha(String _Val) throws Exception {
  MessageDigest myDigest=MessageDigest.getInstance("SHA-256");
  myDigest.update(_Val.getBytes());
  byte[] dataBytes = myDigest.digest();
  StringBuffer sb = new StringBuffer();
  for (int i = 0; i < dataBytes.length; i++) {
    sb.append(Integer.toString((dataBytes[i])).substring(1));
  }
  StringBuffer hexString = new StringBuffer();
  for (int i=0;i<dataBytes.length;i++) {
    String hex=Integer.toHexString(0xff & dataBytes[i]);
    if(hex.length()==1) hexString.append('0');
      hexString.append(hex);
  }
  String retParam = hexString.toString();
  return retParam;
 }
}
```

Java + Hachage sha256 (patch pour v11)

```
SQL> --nous sommes dans sql*plus
CREATE OR REPLACE FUNCTION hash_sha256 (txt varchar2)
RETURN VARCHAR2
AS
LANGUAGE JAVA
NAME 'calcsha256.fsha(java.lang.String) return String';

SQL> select hash_sha256('123456789') from dual;
```

Statistique

- Pour que l'optimisation (Optimizer) puise avoir un plan dit proche optimal, il faut que les statistiques soient à jour. Dans la version 11g R1 + les statistiques étendues sont disponibles. Celles-ci aident pour le calcul (estimation) de la cardinalité.
- Pour l'activer (le DS Dynamic Sampler) il est nécessaire de changer un paramètre de session :

```
SQL> --nous sommes dans sql*plus
ALTER SESSION SET optimizer_dynamic_sampling=4;


-- pour les expressions complexes avec plusieurs prédicats
-- sur la même table avec des (AND OR)
-- le default est 2 valeur possible [0 .. 11]
```

Résumé

- Questions ?
- Nous avons terminé.
- Merci pour votre participation.