Aspects Physiques

S. Lopes, complété par L.Yeh

February 19, 2021

SECTION Les Index

- Introduction
- Types de Requêtes
- Clé d'un index
- Types d'Index
- Index éparses/denses
- Index Primaire/Secondaire
- Index Composite
- Index et Contraintes d'Intégrité
- Maintenance des Index
- Index et Jointure
- Quelques Règles
- Index sous Oracle
- Sélection d'Index

Introduction

- Un index est une structure de données permettant à certaines requêtes d'accéder à un ou plusieurs tuples rapidement
  $\rightarrow$ Le choix, la maintenance et l'utilisation d'index est essentiel pour obtenir de bonnes performances

Introduction

- Un index est une structure de données permettant à certaines requêtes d'accéder à un ou plusieurs tuples rapidement
  $\rightarrow$ Le choix, la maintenance et l'utilisation d'index est essentiel pour obtenir de bonnes performances
- Une mauvaise sélection d'index conduit à
  - des index maintenus mais inutilisés
  - des tables parcourues entièrement pour obtenir un seul tuple
  - des jointures inefficaces
- L'efficacité des index dépend de l'utilisation qu'en font les requêtes

Types de Requêtes

- Requête point: un seul résultat, critère d'égalité

Example 1

```
SELECT name FROM emp WHERE id = 12;
-- id est une clé de emp
```

Types de Requêtes

- Requête point: un seul résultat, critère d'égalité

Example 1

```
SELECT name FROM emp WHERE id = 12;
-- id est une clé de emp
```

- Requête multi-point: plusieurs résultats, critère d'égalité

Example 2

```
SELECT name FROM emp WHERE dept = 'administration';
```

Types de Requêtes

- Requête point: un seul résultat, critère d'égalité

Example 1

```
SELECT name FROM emp WHERE id = 12;
-- id est une clé de emp
```

- Requête multi-point: plusieurs résultats, critère d'égalité

Example 2

```
SELECT name FROM emp WHERE dept = 'administration';
```

- Requête intervalle: plusieurs résultats, inégalité

Example 3

```
SELECT name FROM emp WHERE sal >= 2000;
```

Types de Requêtes

- **Requête préfixe**

Example 4

```
-- Séquence lastname, firstname, city
SELECT sal FROM emp WHERE lastname = 'Gates'
AND firstname LIKE 'Bi%';
```

Types de Requêtes

- Requête préfixe

Example 4

```
-- Séquence lastname, firstname, city
SELECT sal FROM emp WHERE lastname = 'Gates'
AND firstname LIKE 'Bi%';
```

- Requête extrémale: requête sur un préfixe d'une séquence d'attributs

Example 5

```
SELECT name FROM emp
WHERE sal = MAX(SELECT sal FROM emp);
```

Types de Requêtes

- Requête préfixe

Example 4

```
-- Séquence lastname, firstname, city
SELECT sal FROM emp WHERE lastname = 'Gates'
AND firstname LIKE 'Bi%';
```

- Requête extrémale: requête sur un préfixe d'une séquence d'attributs

Example 5

```
SELECT name FROM emp
WHERE sal = MAX(SELECT sal FROM emp);
```

- Requête tri

Example 6

```
SELECT * FROM emp ORDER BY sal;
```

Types de Requêtes

- Requête de regroupement

Example 7

```
SELECT dept, AVG(sal) FROM emp GROUP BY dept;
```

Types de Requêtes

- Requête de regroupement

Example 7

```
SELECT dept, AVG(sal) FROM emp GROUP BY dept;
```

- Requête de jointure

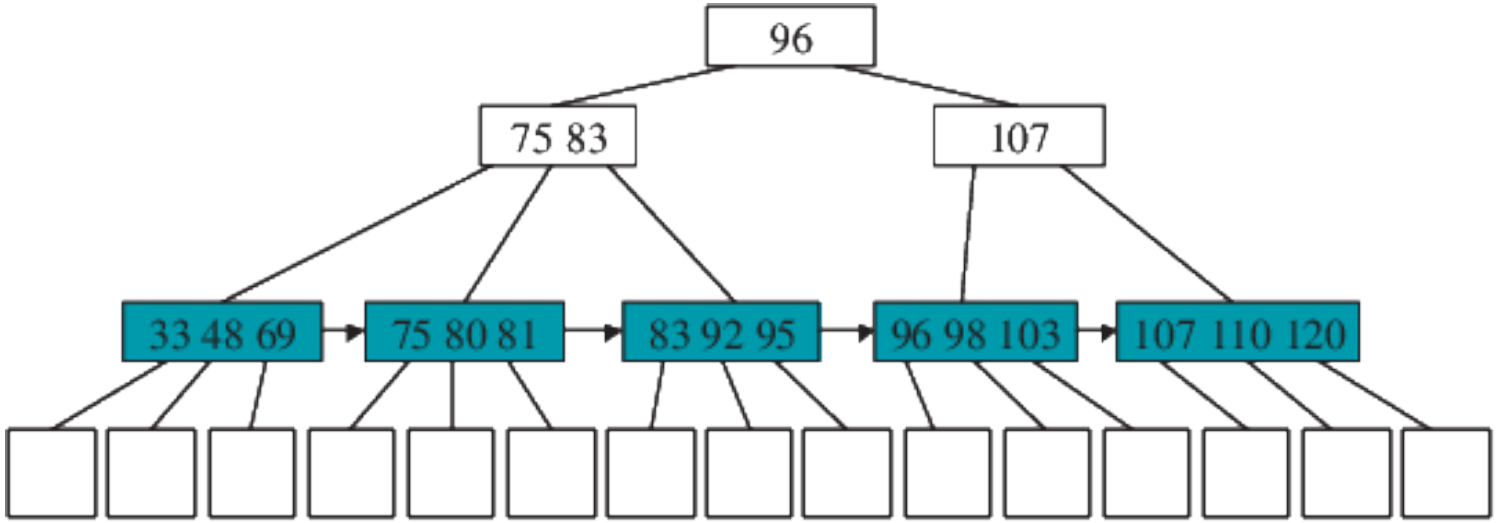
Example 8

```
SELECT emp.name, dept.name FROM emp, dept
WHERE emp.depnum = dept.num;
```

Clé d'un index

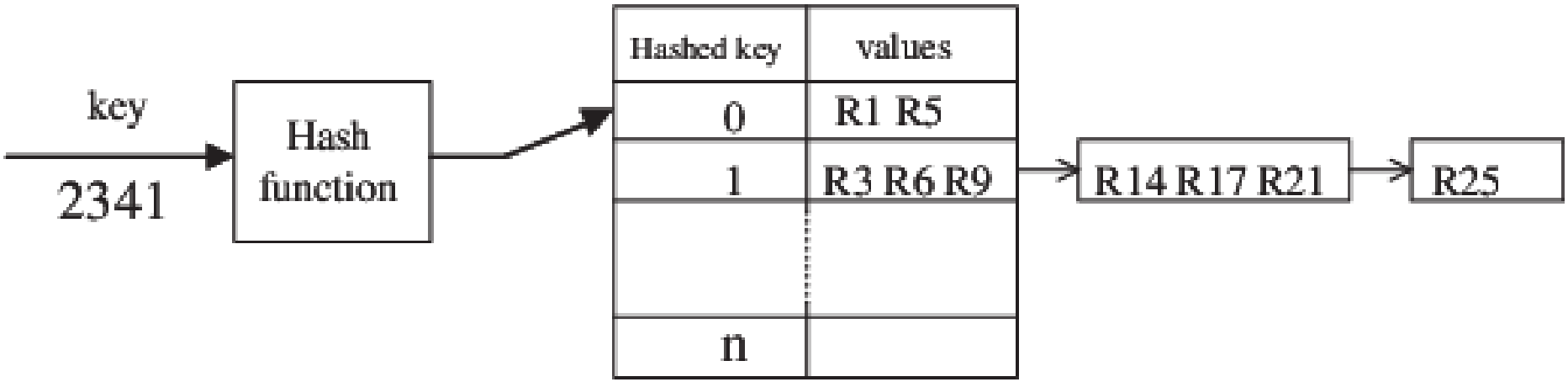
- La clé d'un index est la **séquence** d'attributs servant pour les recherches dans l'index
- Plusieurs tuples peuvent avoir la même valeur de clé (ne pas confondre avec une clé dans le modèle relationnel)
- Deux types de
  - clé séquentielle (monotone par rapport à l'ordre d'insertion)
  - clé non séquentielle (non corrélée avec l'ordre d'insertion)
- Une *clé séquentielle* peut poser des problèmes de performance (contrôle de concurrence)

Arbre B+



- Structure efficace et polyvalente
- Supporte différents types de requêtes: intervalle, extrémal, . . .
- En cas d'insertions fréquentes et de clé *non séquentielle*, on obtient de meilleur temps de réponse qu'avec un index hashé
- La taille de la clé est importante: quand la taille de la clé augmente, le nombre de pointeurs par nœuds diminue
- Paramètres importants: nb de niveaux, nb de fils par nœuds (*fanout*)

Table de hashage



- Structure très efficace mais moins polyvalente
- Permet de répondre à une requête de type point en un accès disque (meilleur structure dans ce cas)
- Adapté pour une requête multi-point si l'index est *plaçant*
- Inadapté pour les requêtes de type interval, extrémal ou préfixe
- Peut devenir inefficace en cas de dépassement
- Taille de la clé n'a pas d'importance (résultat=fonction de hashage)

Autres

- Il existe d'autres types d'index plus spécialisés: bitmap, ...

Index éparses/denses

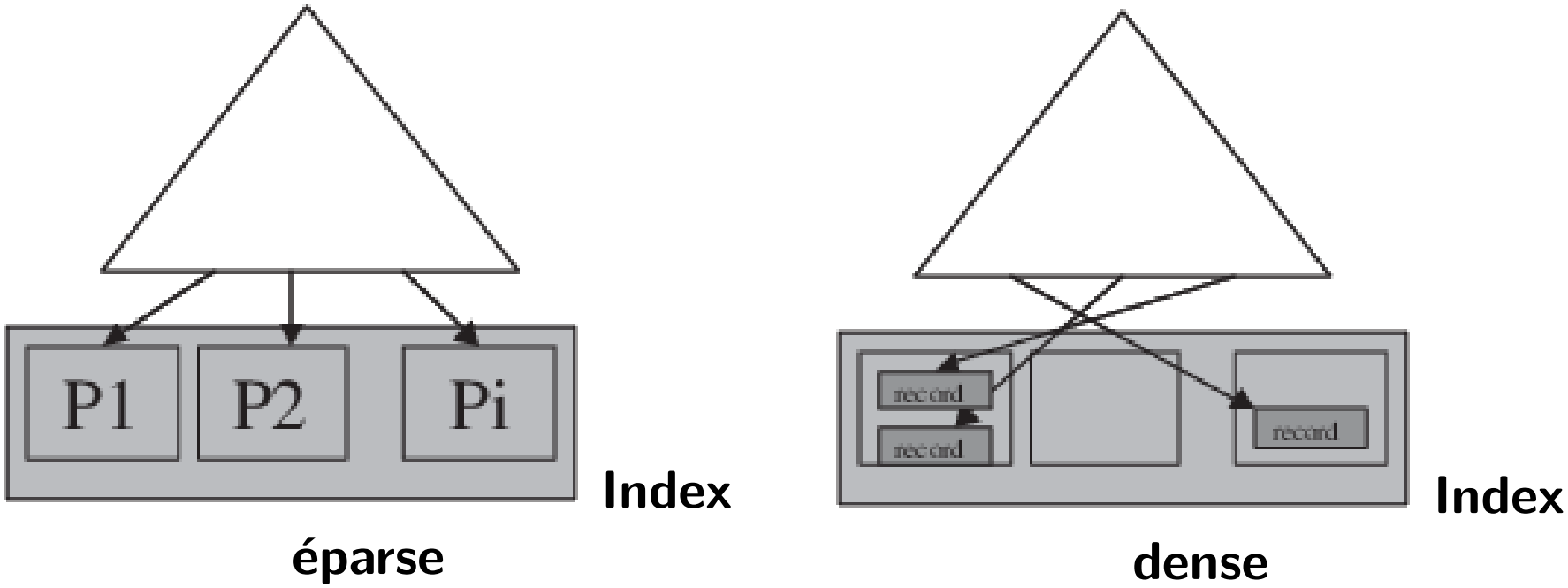


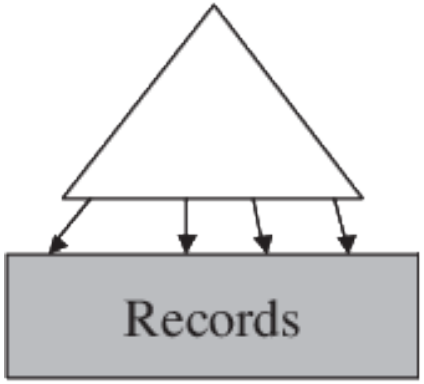
Figure: Index éparses/denses

- Un index éparse possède au plus un pointeur de la structure vers chaque page (figure 1)
- Un index dense possède un pointeur par tuple

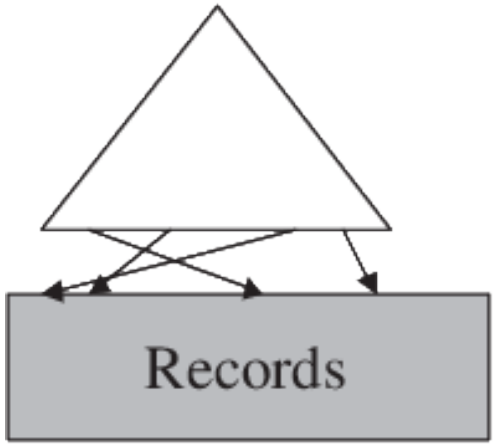
Index éparses/denses

- Un index éparse contient moins de clé qu'un index dense: $NbPtrIdxDense = NbEnrParPage \times NbPtrIdxEparse$
- Si la taille d'un tuple est petite par rapport à la taille d'une page, un index éparse est avantageux (peut avoir un niveau de moins) sinon (taille d'un tuple voisine de la taille d'une page) un index dense est préférable
- Un index dense permet d'exécuter certaines requêtes sans accéder aux données (comptage par exemple). On dit que l'index *couvre* la requête.

Index Primaire/Secondaire



Index primaire



Index secondaire

- Dans un index primaire ou plaçant, l'organisation des données suit l'ordre de l'index
- Un index plaçant peut être dense ou éparse
- On peut créer au plus un index plaçant par table

Index Primaire/Secondaire

- Dans un index *secondaire* ou *non plaçant*, l'organisation des données n'a pas de lien avec l'ordre de l'index
- Un index secondaire est toujours dense
- On peut créer plusieurs index secondaires par table
- Avantages et inconvénients d'un index primaire
  - s'il est éparse (moins de pointeurs), il peut permettre d'économiser un accès (un niveau de moins dans l'arbre)
  - il est très efficace pour les requêtes *multi-points*
  - il permet de répondre aux requêtes intervalles et de tri s'il est basé sur un arbre B+
  - son bénéfice diminue si y a beaucoup de pages de dépassements (réorganisation nécessaire)

Index Primaire/Secondaire: Avantages et inconvénients d'un index secondaire

- il peut éviter d'accéder à la table elle-même (on dit que l'index *couvre* la requête)

Example 9 *Soit un index sur A, B, C et la requête* `SELECT B, C FROM R WHERE A = 5;`

- si plusieurs tables sont critiques, on peut créer plusieurs index pour couvrir les différentes requêtes (similaire à des tables redondantes)
- avec ce type d'index, le nombre de pages accédées est égale au nombre de tuples lus

$\Rightarrow$ idéal si l'index couvre la requête

$\Rightarrow$ bon si le nombre de tuples lus est très inférieur au nombre de pages de la table (pour des requêtes points par exemple)

$\Rightarrow$ dans les autres cas, un scan complet est plus efficace

Index Primaire/Secondaire

Example 10

- *1 page = 4Ko, 1 tuple = 50 octets, A possède 20 valeurs distinctes*
- *une requête accède à $\frac{1}{20}$ des tuples*
- *une page contient 80 tuples donc chaque page contiendra probablement un tuple pour chaque valeur de A*

$\Rightarrow$ *un scan sera plus efficace qu'un accès par un index secondaire*

Example 11

- *même situation mais 1 tuple = 2Ko*
- *une page contient 2 tuples donc la requête touche environ une page sur 10*

$\Rightarrow$ *l'index secondaire sera utile*

Index Composite

- **Définition**: La clé de l'index est formée de plusieurs attributs
- L'ordre des attributs est important: on place en premier l'attribut le plus contraint dans les requêtes

Index Composite

- **Définition**: La clé de l'index est formée de plusieurs attributs
- L'ordre des attributs est important: on place en premier l'attribut le plus contraint dans les requêtes
- Avantages
  - s'il est dense, il permet de répondre à certaines requêtes sans accéder aux données
  - il permet d'assurer l'unicité d'un ensemble de valeurs d'attributs

Index Composite

- **Définition**: La clé de l'index est formée de plusieurs attributs
- L'ordre des attributs est important: on place en premier l'attribut le plus contraint dans les requêtes
- Avantages
  - s'il est dense, il permet de répondre à certaines requêtes sans accéder aux données
  - il permet d'assurer l'unicité d'un ensemble de valeurs d'attributs
- Inconvénients
  - la clé est en général de grande taille donc l'index est encombrant
  - la mise à jour d'un seul des attributs de l'index provoque la mise à jour de l'index

Index et Contraintes d'Intégrité

- Un index secondaire est implicitement créé pour assurer les contraintes d'intégrité `PRIMARY KEY` et `UNIQUE`
- Aucun index n'est automatiquement créé pour les contraintes de type `FOREIGN KEY`

Index et Contraintes d'Intégrité

- Un index secondaire est implicitement créé pour assurer les contraintes d'intégrité `PRIMARY KEY` et `UNIQUE`
- Aucun index n'est automatiquement créé pour les contraintes de type `FOREIGN KEY`
- Les attributs impliqués dans une contrainte référentielle sont de bons candidats pour la création d'un index

Maintenance des Index

- Régulièrement, il est nécessaire de restructurer l'index à cause des débordements (suppression puis reconstruction ou processus spécifique)
- Les statistiques de l'optimiseur doivent être conservées à jour
- Le plan d'exécution des requêtes doit être étudié
- Il est possible de supprimer un index le temps d'exécuter une requête (suppression, exécution, recréation)

Index et Jointure

- un algorithme de jointure par boucle imbriquée est proportionnel à $|R| \times |S|$
- un algorithme de jointure par boucle imbriquée indexé est proportionnel à $|R| \times log(|S|)$
- l'index peut couvrir la requête de jointure

Index et Jointure

- un algorithme de jointure par boucle imbriquée est proportionnel à $|R| \times |S|$
- un algorithme de jointure par boucle imbriquée indexé est proportionnel à $|R| \times log(|S|)$
- l'index peut couvrir la requête de jointure
- dans le cas d'une clé étrangère $R.A \subseteq S.B$, un index sur $S.B$ permet des insertions plus rapides dans R alors qu'un index sur $R.A$ permet des suppressions plus rapides dans S

Quelques Règles

1. Préférer les structures de hash pour les égalités
2. Préférer un index plaçant si la requête a besoin de tous les attributs ou presque
3. Utiliser un index dense pour couvrir les requêtes critiques
4. Ne pas utiliser d'index si le temps perdu en mise à jour excède le gain
5. Eviter les index sur de petites tables (dépend en fait du rapport taille d'un tuple sur taille d'une page)

Index

- La dénomination index sous Oracle correspond à des index secondaires
- Différents types
  - arbre B+
  - bitmap
  - basé sur une fonction
  - ...

Syntaxe:

```
CREATE [BITMAP] INDEX idx ON r(a [ASC|DSC], ...)
[COMPUTE STATISTICS] [COMPRESS] [REVERSE]
```

- `COMPUTE STATISTICS`: calcul en même temps les statistiques pour l'optimiseur
- `COMPRESS`: compresse les clés
- `REVERSE`: inverse l'ordre des octets de la clé

Table organisée selon un index (*index organizad table*)

- Cette notion permet de créer des index primaires
- L'index créé est un arbre B+ sur la clé primaire de la table

Syntaxe:

```
CREATE TABLE r(...) ORGANIZATION INDEX
```

Cluster

- La notion de cluster est destinée à organiser physiquement les données d'une ou plusieurs tables par rapport aux valeurs d'attributs communs aux tables
- Utilisée avec une seule table, elle permet de créer un index primaire hashé
- Principe
  1. Création du cluster
  2. Création de la table dans le cluster
  3. Création d'un index sur le cluster (pour les arbres B+ uniquement)

Syntaxe pour un arbre B+:

```
CREATE CLUSTER pers_clu(nom CHAR(10));
CREATE TABLE personne(nom_pers CHAR(10), adresse INTEGER)
   CLUSTER pers_clu(nom_pers);
CREATE INDEX pers_clu_idx ON CLUSTER pers_clu;
...
DROP CLUSTER pers_clu INCLUDING TABLES;
```

Cluster

Syntaxe pour une table de hashage:

```
CREATE CLUSTER pers_clu(id NUMBER) SINGLE TABLE HASHKEYS 1000;
CREATE TABLE ...
```

- La clé doit être numérique dans le cas d'une table de hashage
- `SINGLE TABLE` permet d'optimiser le cluster pour une seule table
- La clause `HASH IS` permet de préciser la fonction de hashage

Plan d'exécution

- La commande `EXPLAIN PLAN` permet de générer le plan d'exécution pour une requête
- Cette commande remplit une table (`PLAN_TABLE` créé par le script `UTLXPLAN.SQL`) avec des tuples décrivant le plan d'exécution
- En mode optimisation par coût, cette commande estime également le coût d'exécution de la requête

Example 12

```
-- Changement du mode de l'optimiseur
ALTER SESSION SET OPTIMIZER_MODE = CHOOSE;
-- L'optimiseur choisit en fonction de la présence de statistiq
ALTER SESSION SET OPTIMIZER_MODE = RULE; -- Rule based
ALTER SESSION SET OPTIMIZER_MODE = ALL_ROWS;
-- Cost based: optimise le rendement
-- (minimum de ressources pour répondre entièrement)
ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS;
-- Cost based: optimise le temps de réponse
-- (minimum de ressources pour sortir le premier tuple)
```

Plan d'exécution

Syntaxe:

```
EXPLAIN PLAN [SET STATEMENT_ID='...'] FOR requete
```

- La visualisation des résultats se fait en exécutant une requête hiérarchique sur la table `PLAN_TABLE` ou en utilisant le script `UTLXPLS.SQL`

Maintenance des Statistiques

- Les statistiques sont conservées dans le dictionnaire de données
- La méthode utilisée est basée sur des histogrammes de hauteurs voisines (même nombre de valeurs par histogramme)
- Les statistiques sont utiles à l'optimiseur dans le cas d'attributs dont les valeurs ne sont pas uniformément distribuées
- Collecte des statistiques
  - Package `DBMS_STATS`: génération et gestion des statistiques
  - Clause `COMPUTE STATISTICS` de `CREATE INDEX` ou de `ALTER INDEX ... REBUILD`
  - Instruction `ANALYSE` (dépréciée au profit du package)

Example 13

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS('comptoir', 'clients');
-- Collecter des stats pour une table
EXECUTE DBMS_STATS.DELETE_TABLE_STATS('comptoir', 'clients');
-- Supprimer les stats d'une table
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('comptoir');
-- Collecter des stats pour tous les objets d'un schema
```

Maintenance des statistiques

Example 14

```
TABLE_NAME                       NUM_ROWS     BLOCKS AVG_ROW_LEN
------------------------------ ---------- ---------- -----------
ACTORS                             6727         55          52
CASTS                             45442        340          51
MOVIES                            11405        115          68
PEOPLE                             3304         25          43
REMAKES                            1188         15          58
STUDIOS                             195          2          52

EXPLAIN PLAN FOR SELECT title FROM casts WHERE actor = 'Emilio Estevez';
@UTLXPLS
-----------------------------------------------------------------------
| Operation                      |  Name    |  Rows | Bytes|  Cost   |
-----------------------------------------------------------------------
| SELECT STATEMENT               |          |     3 |   96 |      4 |
|  TABLE ACCESS BY INDEX ROW|CASTS     |     3 |   96 |      4 |
|   INDEX RANGE SCAN         |IDX       |     3 |      |      1 |
-----------------------------------------------------------------------
```

Maintenance des statistiques

Example 15

```
EXPLAIN PLAN FOR SELECT /*+ FULL(casts) */ title FROM casts
WHERE actor = 'Emilio Estevez';
@UTLXPLS
--------------------------------------------------------------------
| Operation                 |  Name    |  Rows | Bytes|  Cost  |
--------------------------------------------------------------------
| SELECT STATEMENT          |          |     3 |   96 |     52 |
|  TABLE ACCESS FULL        |CASTS     |     3 |   96 |     52 |
--------------------------------------------------------------------

EXPLAIN PLAN FOR SELECT title FROM casts WHERE actor <> 'Emilio Estevez';
@UTLXPLS
--------------------------------------------------------------------
| Operation                 |  Name    |  Rows | Bytes|  Cost  |
--------------------------------------------------------------------
| SELECT STATEMENT          |          |   45K|    1M|     52 |
|  TABLE ACCESS FULL        |CASTS     |   45K|    1M|     52 |
--------------------------------------------------------------------
```

Maintenance des statistiques

Example 16

```
EXPLAIN PLAN FOR SELECT /*+ INDEX(casts idx) */ title FROM casts
WHERE actor <> 'Emilio Estevez';
@UTLXPLS
--------------------------------------------------------------------
| Operation                    |  Name    |  Rows | Bytes|  Cost  |
--------------------------------------------------------------------
| SELECT STATEMENT             |          |    45K|    1M|  40068 |
|  TABLE ACCESS BY INDEX ROW|CASTS     |    45K|    1M|  40068 |
|   INDEX FULL SCAN          |IDX       |    45K|      |    154 |
--------------------------------------------------------------------
```

Sélection d'Index

- Etape importance du tuning physique d'une BD
- Nécessite de connaître un *workload* (charge de requêtes) caractéristique du système ainsi que les besoins de l'utilisateur
- Charge de requêtes
  - ensemble de requêtes (sélection et mise à jour) avec leurs fréquences
  - pour chaque requête
    - relations concernées
    - attributs concernées (`SELECT`)
    - attributs avec condition (`WHERE`) et leur sélectivité
- Choix de la clé
  - les attributs impliqués dans le `WHERE` sont candidats
  - une égalité sugère un index hashé, un intervalle un arbre B+

Sélection d'Index

- **Clé composite**
  - si plusieurs attributs d'une même relation dans le `WHERE`
  - pour faire en sorte que l'index couvre la requête
  - attention à l'ordre des attributs pour la clé

Sélection d'Index

- **Clé composite**
  - si plusieurs attributs d'une même relation dans le `WHERE`
  - pour faire en sorte que l'index couvre la requête
  - attention à l'ordre des attributs pour la clé
- **Primaire ou non**
  - les requêtes intervalles bénéficient le plus d'un index plaçant
  - si plusieurs alternatives sont envisageables, considérer la fréquence des requêtes et la sélectivité des critères
  - si un index couvre une requête, il est inutile qu'il soit plaçant

Sélection d'Index

- **Clé composite**
  - si plusieurs attributs d'une même relation dans le `WHERE`
  - pour faire en sorte que l'index couvre la requête
  - attention à l'ordre des attributs pour la clé
- **Primaire ou non**
  - les requêtes intervalles bénéficient le plus d'un index plaçant
  - si plusieurs alternatives sont envisageables, considérer la fréquence des requêtes et la sélectivité des critères
  - si un index couvre une requête, il est inutile qu'il soit plaçant
- **Arbre B+ ou hash**
  - un arbre B+ sera en mesure de supporter plus de types de requêtes
  - une table de hashage si une requête impliquant un critère d'égalité est importante

Sélection d'Index

- **Clé composite**
  - si plusieurs attributs d'une même relation dans le `WHERE`
  - pour faire en sorte que l'index couvre la requête
  - attention à l'ordre des attributs pour la clé
- **Primaire ou non**
  - les requêtes intervalles bénéficient le plus d'un index plaçant
  - si plusieurs alternatives sont envisageables, considérer la fréquence des requêtes et la sélectivité des critères
  - si un index couvre une requête, il est inutile qu'il soit plaçant
- **Arbre B+ ou hash**
  - un arbre B+ sera en mesure de supporter plus de types de requêtes
  - une table de hashage si une requête impliquant un critère d'égalité est importante
- **Considérer les mises à jour**
  - une fois créée la liste d'index candidats
  - si un index ralentit une mise à jour, considérer sa suppression
  - attention un index peut aussi accélérer une mise à jour

Outils pour assister la sélection d'index

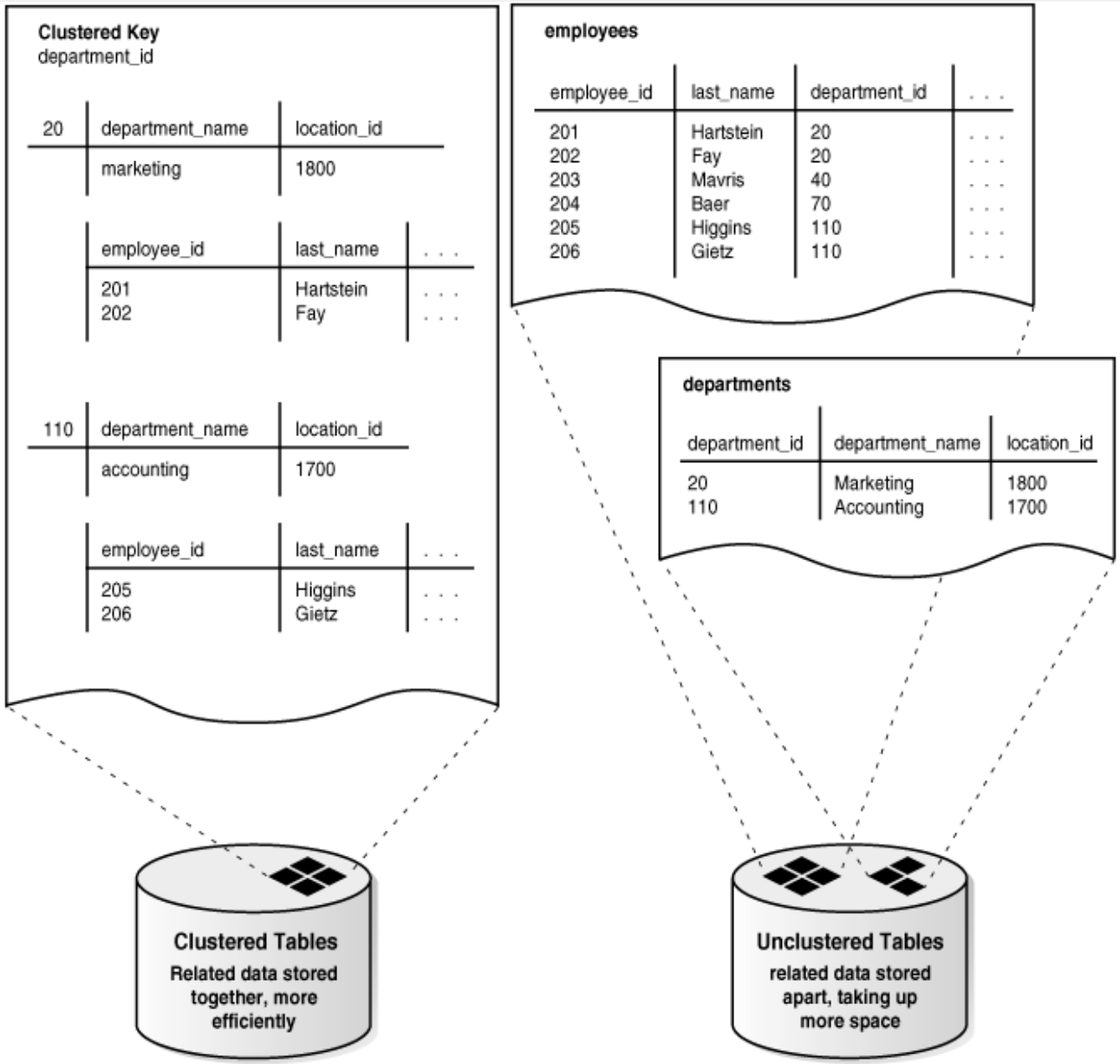
- Le nombre d'index possibles est très grand
  - pour une relation, tous les sous-ensembles d'attributs
  - l'ordre des attributs a de l'importance
  - primaire ou non
  - les grosses applications peuvent manipuler plusieurs milliers de tables

  $\Rightarrow$ outils pour la sélection automatique d'index à partir d'un workload
- Une configuration d'index est un ensemble d'index pour un schéma
- Le coût d'une configuration est le coût total du workload pour cette configuration
- L'objectif est de trouver une configuration de coût minimal (ou au moins une bonne configuration)
- Premier outil: Microsoft SQL Server
- L'efficacité de l'outil dépend
  - du nombre de configurations candidates à tester
  - du nombre d'appel à l'optimiseur pour évaluer les configurations

- L'évaluation du coût est problématique car il est impossible de créer la configuration pour la tester $\Rightarrow$ adaptation de l'optimiseur pour manipuler des index virtuels
- Sélection des candidats
  - heuristique
    1. analyser chaque requête individuellement
    2. faire l'union des index
  - une méthode simple consiste à énumérer tous les index possibles de moins de k attributs (coûteux mais exhaustif)
  - d'autres stratégies sont possibles

Regroupements de Plusieurs Tables

- Regroupement des tuples de plusieurs tables selon les valeurs d'un attribut commun
- **But**: placer les données à joindre proches les unes des autres
- Notion notamment supportée par Oracle (*cluster*)



Regroupements de Plusieurs Tables

- **Avantages**
  - les requêtes sur la clé du cluster (jointures par exemple) sont rapides (par exemple, retrouver les employés d'un département)
  - les requêtes de type point sur les tables du cluster utilisant un index fonctionnent de la même façon qu'avec des index secondaires classiques

Regroupements de Plusieurs Tables

- **Avantages**
  - les requêtes sur la clé du cluster (jointures par exemple) sont rapides (par exemple, retrouver les employés d'un département)
  - les requêtes de type point sur les tables du cluster utilisant un index fonctionnent de la même façon qu'avec des index secondaires classiques
- **Inconvénients**
  - un scan complet des employés sera un peu plus coûteux
  - un scan complet des départements sera beaucoup plus coûteux
  - les débordements suite aux insertions peuvent dégrader les performances

Regroupements de Plusieurs Tables

Example 17

- *Création d'un cluster basé sur un arbre B+*

```
CREATE CLUSTER emp_dept (deptno NUMBER);
CREATE TABLE dept (...) CLUSTER emp_dept(dno);
CREATE TABLE emp (...) CLUSTER emp_dept(depnum);
CREATE INDEX emp_dept_idx ON CLUSTER emp_dept;
```

- *Création d'un cluster basé sur une table de hashage*

```
CREATE CLUSTER emp_dept (deptno NUMBER)
HASH IS deptno HASHKEYS 150;
CREATE TABLE ...
```

Données du Problème

Soit le schéma suivant (les clés sont soulignées, les clés étrangères sont en italiques):
`emp(`<u>`eno`</u>`, ename, age, adr, sal, `*`dno`*`, comm)`
`dept(`<u>`dno`</u>`, dname, `*`mgr`*`, comm)`
La table `emp` comporte 10000 tuples, la table `dept` 50. Une page de données peut contenir 30 tuples de la table `emp`. L'attribut `emp.sal` possède 5 valeurs distinctes: `20000, 25000, 30000, 40000, 48000`. Les valeurs de `emp.age` sont uniformément distribuées entre 20 et 70.

Exercices

Exercice 1 *Soit la requête:* `SELECT * FROM emp WHERE ename = 'dupond';`. *Un index secondaire existe sur* `ename` *mais il n'est pas utilisé par l'optimiseur. Pourquoi ?*

Exercice 2 *Soit la requête:* `SELECT * FROM emp WHERE sal/12 = 4000;`. *Un index secondaire existe sur* `sal` *et les statistiques sont à jour. Cependant, l'index n'est pas utilisé. Pourquoi ?*

Exercice 3 *On a modifié la requête précédente comme conseillé mais l'index n'est toujours pas utilisé. Pourquoi ?*

Exercices

Exercice 4 *Un index secondaire sur `emp.dno` est-il utile pour des requêtes multi-points sur `emp.dno` ?*

Exercice 5 *Supposons que l'on dispose de 5000 départements. Qu'en est-il alors ?*

Exercices

Exercice 6 *Soit la requête:* `SELECT * FROM emp WHERE dno = 5;`. *On peut stocker 250 pointeurs dans une page d'index. Combien de pages seront lues pour répondre à cette requête dans les cas suivants:*

1. *sans index,*
2. *avec un index primaire B+ de 3 niveaux,*
3. *avec un index secondaire B+ de 3 niveaux,*
4. *avec un index primaire hashé,*
5. *avec un index secondaire hashé.*

Exercices

Dans l'exercices suivant, proposer une sélection d'index améliorant la ou les requêtes.

Exercice 7

```
SELECT ename FROM emp WHERE eno = 3
```

Exercices

Dans l'exercices suivant, proposer une sélection d'index améliorant la ou les requêtes.

Exercice 8

```
SELECT * FROM emp WHERE dno = 5
```

Exercices

Exercice 9

```
SELECT ename FROM emp WHERE sal BETWEEN 25000 AND 40000
```

Exercice 10

```
SELECT emp.ename, dept.mgr FROM emp, dept
WHERE dept.dname = 'ventes' AND emp.dno = dept.dno
```

Exercice 11 *On reprends la requête de l'exercice précédent en ajoutant `AND emp.age = 25`.*

Exercice 12

```
SELECT ename, dname FROM emp, dept
WHERE emp.sal BETWEEN 25000 AND 40000
AND emp.adr = 'paris' AND emp.dno = dept.dno
```

Exercices

Exercice 13

```
SELECT dno, COUNT(eno) FROM emp GROUP BY dno
```

Exercice 14

```
SELECT dname, mgr, eno FROM emp, dept
WHERE emp.dno = dept.dno
```

Exercices

Exercice 15 *Considérer maintenant les requêtes des exercices précédents (à partir du 7) comme une charge de requêtes à optimiser.*

Exercices