6 Indexation

Sommaire

- Concepts de base pour l'indexation: index denses, non denses, index multi-niveaux Arbre-B, structure et algorithmes
- Hachage
- Index bitmap
- Exemples avec Oracle

Pourquoi l'indexation

- En l'absence d'un index, seules solutions :
  - Parcours séquentiel (complexité linéaire)
  - Recherche par dichotomie si fichier trié (complexité logarithmique)
- Avec un index:
  - Parcours de l'index, puis accès direct à l'enregistrement
  - Mais attention: mises à jour plus coûteuses !

Exemple 1: une table

titre	année	titre	année
Vertigo	1958	Annie Hall	1977
Brazil	1984	Jurasic Park	1992
Twin Peaks	1990	Metropolis	1926
Underground	1995	Manhattan	1979
Easy Rider	1969	Reservoir Dogs	1992
Psychose	1960	Impitoyable	1992
Greystoke	1984	Casablanca	1942
Shining	1980	Smoke	1995

Exemple 2

La table des films, avec:

- 1000000 (un million) de films
- Un enregistrement = 120 octets
- Un bloc = 4K => 34 enregistrements par bloc
- Environ 30 000 blocs, 120 Mo

Clé de recherche, opérations

Clé de recherche = une liste d'un ou plusieurs colonnes sur lesquels portent des critères.

Types de recherche :

- Recherche par clé : on associe une valeur à chaque attribut de la clé
- Recherche par intervalle : on associe un intervalle de valeurs à chaque attribut de la clé
- Recherche par préfixe : on associe une valeur à un préfixe de la clé

Exemples

Clés de recherche :

- Le titre du film (c'est aussi la clé primaire)
- L'année du film
- Le titre plus l'année

Opérations:

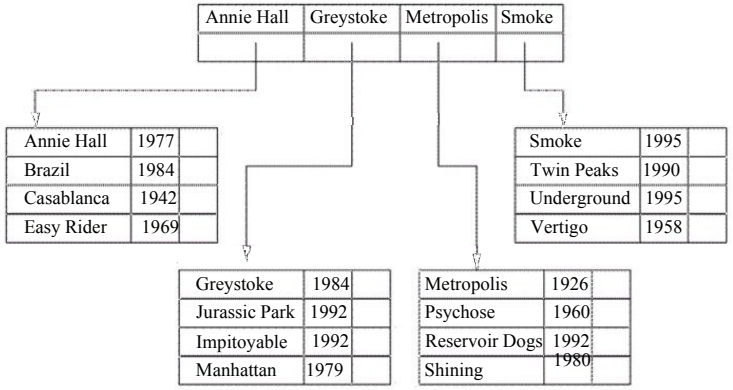
- Rechercher Vertigo
- Rechercher les films parus entre 1960 et 1975
- Rechercher les films commençant par 'V'

Index non-dense

Hypothèse: table triée sur la clé
L'index:

- Les enregistrements (ou entrées) de l'index sont de la forme `[valeur, Addr]`
- L'index est trié sur `valeur`
- Un seul enregistrement par bloc de données est représenté dans l'index

Exemple



Opérations

Recherches:

- Par clé : par dichotomie sur l'index
- Par intervalle : recherche de la borne inférieure, accès au fichier, parcours séquentiel (exemple [J,P])
- Par préfixe : cas particulier de la recherche par intervalle

Exemple concret

Sur notre fichier de 120 Mo

- En supposant qu'un titre occupe 20 octets, une adresse 8 octets
- Taille de l'index : $29142*(20+8) = 815976$ octets

➢ Beaucoup plus petit que la table ! Avantage principal sur la recherche par dichotomie.

Problème: maintenir l'ordre sur la table et sur l'index.

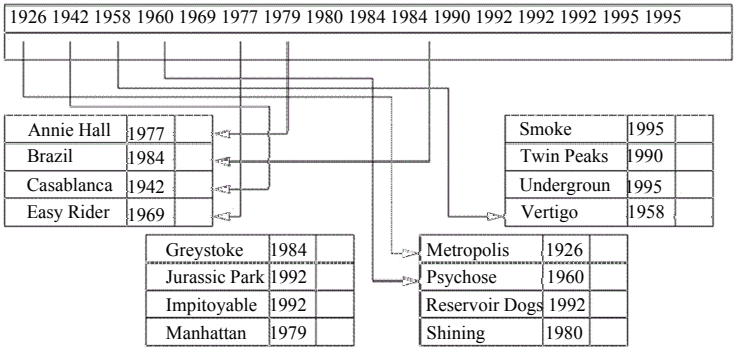
Index dense

Si on veut indexer une table non triée ?

- L'index est toujours trié sur la clé
- Tous les enregistrements sont représentés

L'index est dit dense.

Exemple



Exemple concret

Sur notre fichier de 120 Mo

- Une année = 4 octets; une adresse = 8 octets

Taille de l'index: $1000000*(4+8) = 12$ Mo

Seulement dix fois plus petit que le fichier : la taille d'un index ne doit pas être négligée ici.

Opérations

Recherches:

- Par clé: comme sur un index non-dense
- Par intervalle (exemple [1950, 1979]) :
  - recherche dans l'index de la borne inférieure
  - parcours séquentiel dans l'index
  - à chaque valeur : accès au fichier de données Coût beaucoup plus élevé.

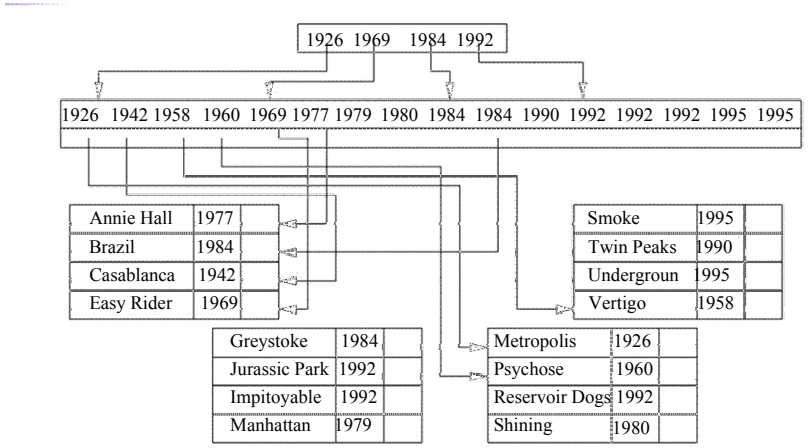
Index multi-niveaux

Si l'index est trop gros ? On l'indexe à son tour.

- Essentiel : l'index est trié, donc on peut l'indexer par un second niveau non-dense Sinon ça ne servirait à rien (pourquoi ?)
- Donc dès le second niveau on diminue drastiquement la taille.
- On peut continuer jusqu'à un niveau avec une seule page.

On obtient une structure séquentielle indexée

Exemple



Remarques / vocabulaire

- Index plaçant ?
  - index qui détermine la position des données
- Ne peut y avoir qu'un index non-dense sur un fichier (pourquoi ?)
- Il peut y avoir autant d'index dense que l'on veut

Arbre-B

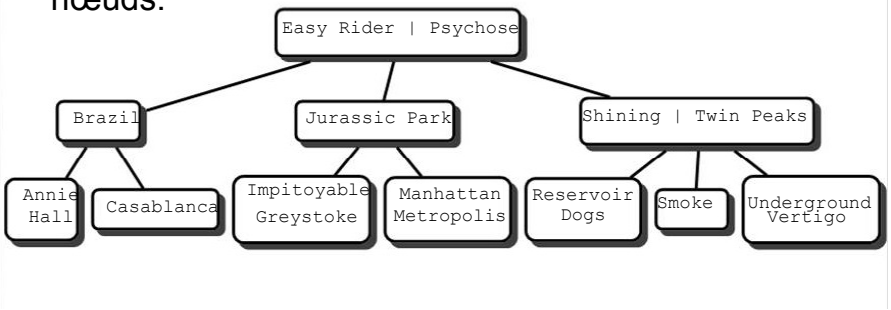
Aboutissement des structures d'index basées sur l'ordre des données

- c'est un arbre équilibré
- chaque nœud est un index local
- il se réorganise dynamiquement
- utilisé universellement !

Comparable aux séquentiel indexé, mais évite d'avoir à maintenir des fichiers triés.

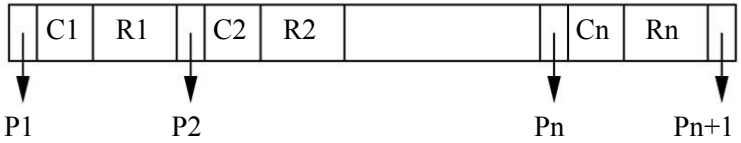
Exemple d'un arbre B

Un arbre, avec trois niveaux, une racine, les enregistrements répartis dans les nœuds.



Les recherches sont guidées par l'ordre dans chaque nœud.

Nœud d'un arbre B



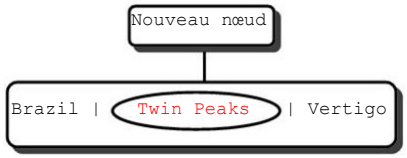
Un nœud est un index local, les enregistrements servant de clé, intercalés avec des pointeurs.

Le sous-arbre pointé par P_2 contient tous les enregistrements dont la clé est comprise entre C_1 et C_2.

Construction de l'arbre-B

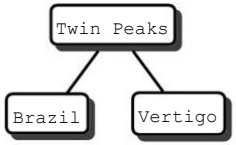
On suppose qu'on ne peut mettre que deux enregistrements par bloc. Voici le premier bloc de l'arbre

Construction de l'arbre-B



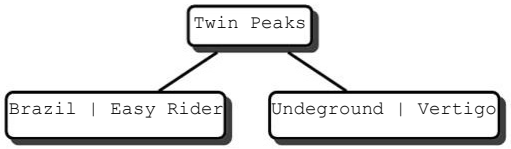
Ici le bloc déborde. On prend l'élément du milieu pour le monter dans un nouveau nœud

Construction de l'arbre-B



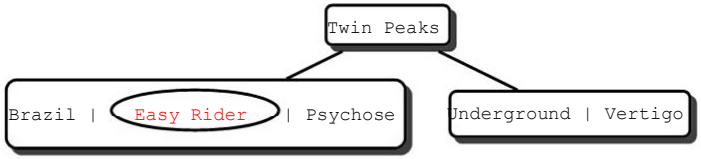
Après répartition des enregistrements

Construction de l'arbre-B



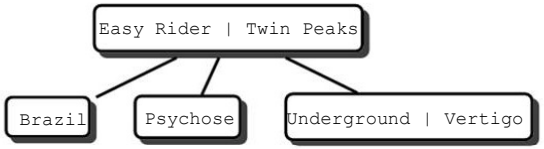
Après insertion de Underground et Easy Rider
Maintenant il faut mettre Psychose

Construction de l'arbre-B



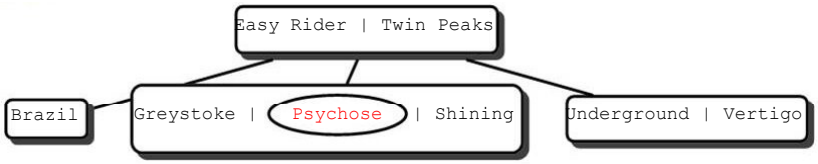
Insertion de Psychose: le nœud déborde, et l'élément du milieu doit monter.

Construction de l'arbre-B



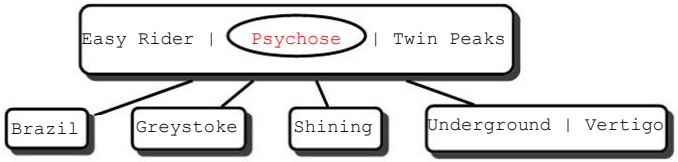
Après remontée de Easy Rider, et répartition des autres enregistrements

Construction de l'arbre-B



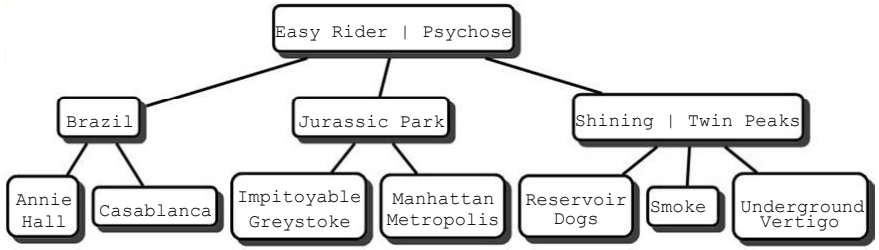
On insère Shining: ça déborde

Construction de l'arbre-B



On a remonté Psychose, mais maintenant c'est la racine qui déborde !!

Construction de l'arbre-B



… et voilà le résultat final

L'arbre B+

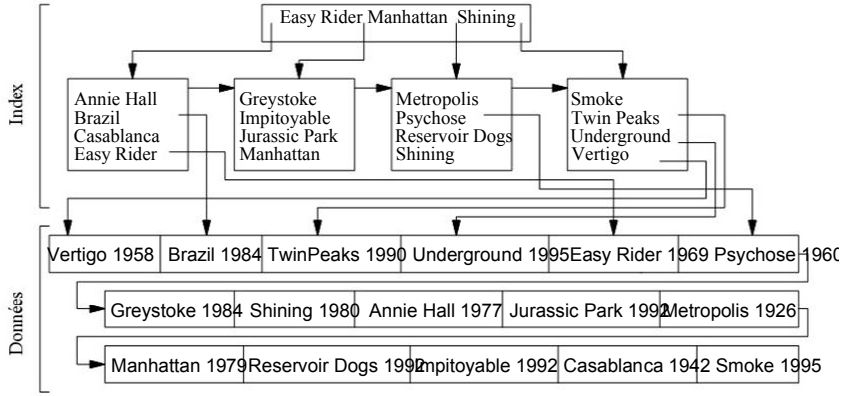
Inconvénient de l'arbre B

- Il est plaçant (un seul par fichier)
- Les enregistrements sont déplacés pendant la construction (pb d'adressage)

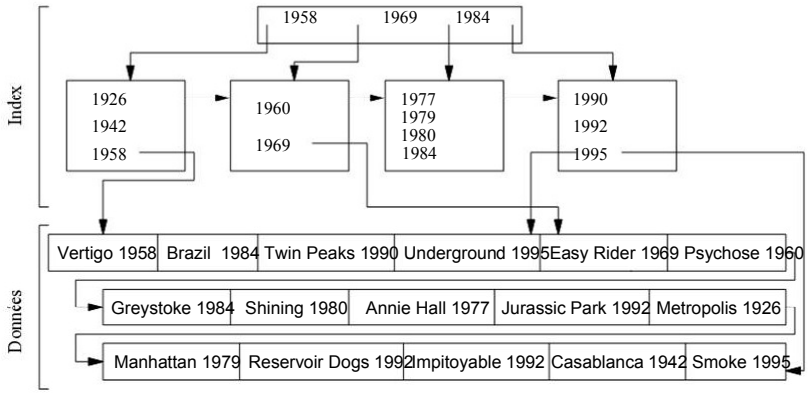
L'arbre B+ est une variante non plaçante

- Construit uniquement sur les clés
- Toutes les clés sont conservées dans les feuilles
- Dans les feuilles, à chaque clé est associée l'adresse de l'enregistrement

Arbre B+ : exemple 1



Arbre B+ : exemple 2



Exemple concret

Fichier de 120 Mo.

- une entrée = 28 octets. Donc 4096/28 = 146 entrées par bloc
- 1000000/146 = 6850 blocs pour le premier niveau de l'arbre-B+
- 6850/146 = 47 blocs pour le second niveau un bloc avec 47 entrées pour le troisième niveau

Recherche par clé

```
SELECT * FROM Film WHERE titre =
'Impitoyable'
```

- on lit la racine de l'arbre: Impitoyable étant situé dans l'ordre lexicographique entre Easy Rider et Manhattan, on doit suivre le chaînage situé entre ces deux titres;
- on lit le bloc feuille dans lequel on trouve le titre Impitoyable associé à l'adresse de l'enregistrement dans le fichier des données;
- il reste à lire l'enregistrement.

Recherche par intervalle

```
SELECT * FROM Film WHERE annee
BETWEEN 1960 AND 1975
```

- On fait une recherche par clé pour l'année 60 On parcourt les feuilles de l'arbre en suivant le chaînage, jusqu'à l'année 1975
- A chaque fois ont lit l'enregistrement

Attention, les accès aux fichiers peuvent coûter très cher.

Recherche par préfixe

Exemple:

```
SELECT * FROM Film WHERE titre LIKE 'M%'
```

- Revient à une recherche par intervalle.

```
SELECT * FROM Film WHERE titre
BETWEEN 'MAAAAAA...' AND
'MZZZZZZ...'
```

Contre-exemple :

```
SELECT * FROM Film WHERE titre LIKE '%e'
```

- Ici index inutilisable

Capacité d'un arbre B

- Avec un niveau d'index (la racine seulement) on peut donc référencer 146 films;
- Avec deux niveaux on indexe 146 blocs de 146 films chacun, soit $146*146 = 21316$ films; avec trois niveaux on indexe $146*146*146 = 3112136$ films;
- Avec quatre niveaux on index plus de 450 millions de films.

L'efficacité d'un arbre-B+ dépend de la taille de la clé : plus elle est petite, plus l'index sera petit et efficace.

Efficacité de l'arbre B+

Il est (presque) parfait !

- On a très rarement besoin de plus de trois niveaux
- Le coût d'une recherche par clé est le nombre de niveaux, plus 1.
- Supporte les recherches par clé, par intervalle, par préfixe
- Dynamique

On peut juste lui reprocher d'occuper de la place.

Le hachage

Le hachage

Un concurrent de l'arbre-B+

- Meilleur pour les recherches par clé
- N'occupe aucune place

Mais

- se réorganise difficilement
- ne supporte pas les recherches par intervalle

Principe du hachage

On calcule la position d'un enregistrement d'après la clé.

- une fonction de hachage h associe des valeurs de clé à des adresses de bloc
- doit répartir uniformément les enregistrements dans les n blocs alloués à la structure
- recherche d'un enregistrement => calcul de l'endroit où il se trouve.

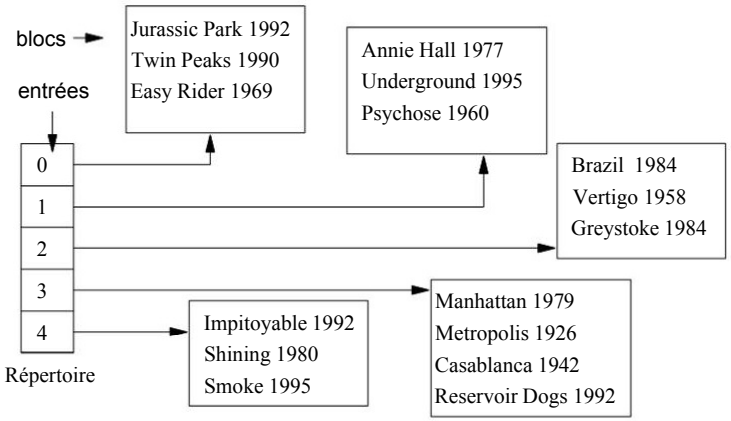
Simple et efficace!

Exemple

On veut créer une structure de hachage pour nos 16 films (hyp. : 4 enregistrements par page)

- On alloue 5 pages (pour garder une marge de manœuvre)
- Un répertoire à 5 entrées (0 à 4) pointe vers les pages
- On définit la fonction
  h(titre) = rang(titre[0])mod5

Le résultat



Recherches

- Par clé, Oui:

```
SELECT * FROM Film WHERE titre
= 'Impitoyable'
```

- Par préfixe ? Ici, oui.

```
SELECT * FROM Film WHERE titre
LIKE 'M%'
```

- Par intervalle: non !

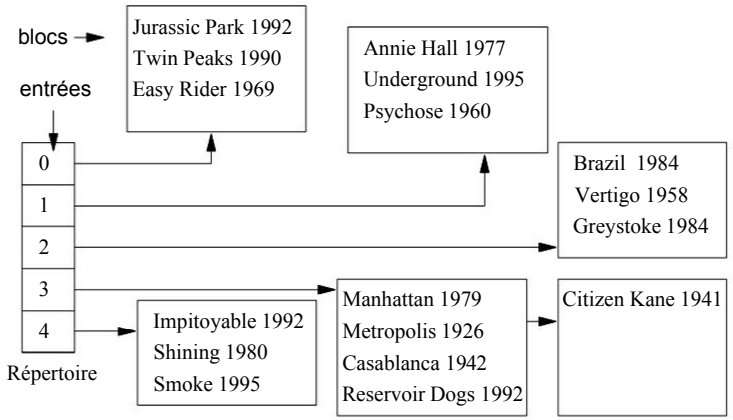
```
SELECT * FROM Film WHERE titre
BETWEEN 'Annie Hall' AND 'Easy
Rider'
```

Mises à jour : ça se gâte

La structure simple décrite précédemment n'est pas dynamique

- On ne peut pas changer un enregistrement de place
- Donc il faut créer un chaînage de pages quand une page déborde
- Et donc les performances se **dégradent**...

Exemple : insertion de Citizen Kane



Hachage dynamique

Objectif: **réorganiser** la table de hachage en fonction des insertions et suppressions.

- le nombre d'entrées dans le répertoire est une puissance de 2
- la fonction h donne toujours un entier sur 4 octets (32 bits)

Idée de base: on utilise les n premiers bits du résultat de la fonction, avec $n < 32$

Exemple : le hachage des 16 films

titre	h(titre)
Vertigo	01110010
Brazil	10100101
Twin Peaks	11001011
Underground	01001001
Easy Rider	00100110
Psychose	01110011
Greystoke	10111001
Shining	11010011

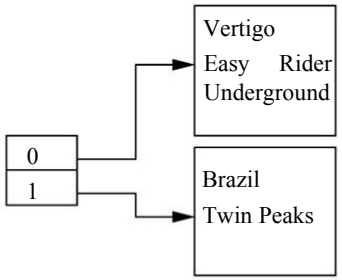
Construction de la table

Au départ on utilise seulement le premier bit de la fonction

- Deux valeurs possibles: 0 et 1
- Donc deux entrées, et deux blocs
- L'affectation d'un enregistrement dépend du premier bit de sa fonction de hachage

=> pour l'instant on reste dans un cadre classique

Avec 5 films



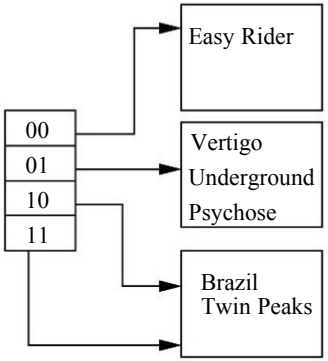
Insertions

Supposons 3 films par bloc. L'insertion de Psychose (valeur 01110011) entraîne le débordement du premier bloc.

- On double la taille du répertoire
- On alloue un nouveau bloc pour l'entrée 01
- Les entrées 10 et 11 pointent sur le même bloc.

On agrandit seulement ce qui est nécessaire.

Illustration



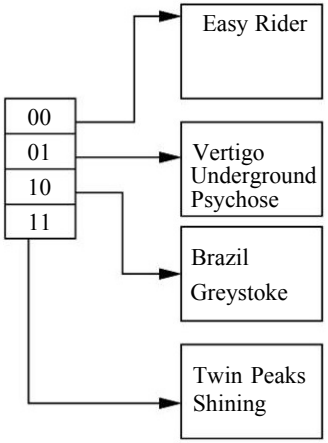
Insertions suivantes

Plusieurs cas

- on insère dans un bloc plein, mais plusieurs entrées pointent dessus
  - on alloue un nouveau bloc, et on répartit les pointeurs
- on insère dans un bloc plein, associé à une entrée
  - on double à nouveau le nombre d'entrées

  Problème: le répertoire peut devenir très grand.

Greystoke 10111001 et Shining 11010011



Comparaison hachage / arbre-B

La hachage, intéressant quand :

- Le jeux de données est figé
- Les recherches se font par clé

Ce sont des situations relativement courantes :
le hachage n'occupe alors pas de place.
Sinon l'arbre-B est meilleur.

Index bitmap

Le problème

Comment indexer une table sur un attribut qui ne prend qu'un petit nombre de valeurs?

- Avec un arbre B : pas très bon car chaque valeur est peu sélective
- Avec un hachage : pas très bon non plus car il y a beaucoup de collisions.

Or situation fréquente, notamment dans les entrepôts de données

Exemple : codification des films

rang	titre	genre
1	Vertigo	Suspense
2	Brazil	Science-Fiction
3	Twin Peaks	Fantastique
4	Underground	Drame
5	Easy Rider	Drame
6	Psychose	Drame
7	Greystoke	Aventures
8	Shining	Fantastique

Principes de l'index bitmap

Soit un attribut A, prenant n valeurs possibles $[v_1,...v_n]$

- On crée n tableaux de bit, un pour chaque valeur v_i
- Ce tableau contient un bit pour chaque enregistrement e
- Le bit d'un enregistrement e est à 1 si e.A = v_i, à 0 sinon

Exemple

	1	2	3	4	5	6	7	8
Drame	0	0	0	1	1	1	0	0
Science-Fiction	0	1	0	0	0	0	0	0
Comédie	0	0	0	0	0	0	0	0

	9	10	11	12	13	14	15	16
Drame	0	0	0	0	0	0	1	0
Science-Fiction	0	1	1	0	0	0	0	0
Comédie	1	0	0	1	0	0	0	1

Recherche

Soit une requête comme:

```
SELECT * FROM Film WHERE genre='Drame'
```

- On prend le tableau pour la valeur `Drame`
- On garde toutes les cellules à 1
- On accède aux enregistrements par l'adresse => très efficace si n, le nombre de valeurs, est petit.

Autre exemple

```
SELECT COUNT(*) FROM Film WHERE genre
IN ('Drame', 'Comédie')
```

- On compte le nombre de 1 dans le tableau `Drame`
- On compte le nombre de 1 dans le tableau `Comédie`
- On fait la somme et c'est fini

=> typique des requêtes DataWarehouse

Indexation dans Oracle

Les choix d'Oracle

Oracle propose à peu près toutes les structures d'index vues précédemment.

- Par défaut l'index est un arbre B+
- Il est possible d'organiser une table en arbre B (plaçant)
- Le hachage (non dynamique) existe aussi
- Les index bitmap

Hachage

Structure appelée `Hash Cluster`. Utilisée en deux étapes

- On crée la structure avec tous ses paramètres
- On affecte une ou plusieurs tables à la structure

Attention, le hachage dans Oracle n'est pas dynamique

Création d'un Hash Cluster

```
CREATE CLUSTER HachFilms (id
INT) SIZE 500 HASHKEYS 500;
```

- La clé de hachage est de type `INTEGER`; Oracle fournit automatiquement une fonction avec de bonnes propriétés
- Nombre de valeurs de la fonction donné par `HASHKEYS`
- Taille de chaque entrée estimée par SIZE
  Donc ici 8 entrées par bloc

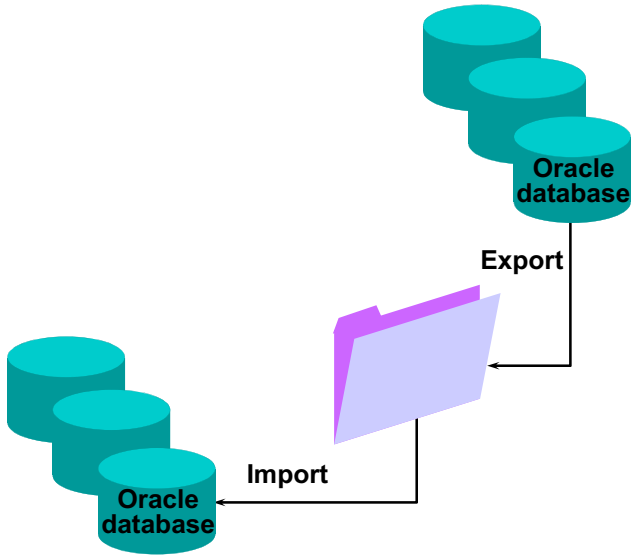
7

Export/Import
RMAN

Objectifs

- Utilitaires Export et Import
- Opérations Export and Import
- RMAN

Oracle Export et Import



Utilitaires Oracle Export et Import

Vous pouvez utiliser ces outils pour les tâches suivantes :

- Archiver les données
- Sauvegarder les définitions des tables pour les protéger des erreurs utilisateurs
- Transférer les données entre machines et bases de données
- Transporter les tablespaces entre bases de données

Modes pour Export

Mode Table	Mode User	Mode Tablespace	Mode Full Database
Table definitions	Tables definitions	Table definitions	Tables definitions
Table data (all or selected rows)	Tables data		Tables data
Owner's table grants	Owner's grants	Grants	Grants
Owner's table indexes	Owner's indexes	Indexes	Indexes
Table constraints	Tables constraints	Table constraints	Tables constraints
		Triggers	

Commandes Export

- **Syntax:**

```
exp keyword = value, value2, … ,valuen
```

- **Examples:**

```
exp hr/hr TABLES=employees,departments
    rows=y file=exp1.dmp
```

```
exp system/manager OWNER=hr
    file=expdat.dmp
```

```
exp \'username/password AS SYSDBA\'
    TRANSPORT_TABLESPACE=y
    TABLESPACES=ts_emp log=ts_emp.log
```

Utilitaire Import pour la Restauration

- Créer les définitions des tables
- Extraire les données à partir d'un fichier Export valide
- Importer à partir d'un fichier Export complet ou cumulatif
- Restaurer après des erreurs utilisateurs

Modes Import

Mode	Description
Table	Import specified tables into a schema.
User	Import all objects that belong to a schema
Tablespace	Import all definitions of the objects contained in the tablespace
Full Database	Import all objects from the export file

Commandes Import

- **Syntax:**

```
imp keyword = value or keyword = value,
value2, … value n
```

- **Examples:**

```
imp hr/hr TABLES=employees,departments
    rows=y file=exp1.dmp
```

```
imp system/manager FROMUSER=hr
file=exp2.dmp
```

```
imp \'username/password AS SYSDBA\'
    TRANSPORT_TABLESPACE=y
    TABLESPACES=ts_employees
```

Commande Import as SYSDBA

- Vous aurez besoin d'invoquer Import as SYSDBA pour importer des tablespaces
- Invoquer Import as SYSDBA:

```
imp \'username/password AS SYSDBA\'
```

Processus Import

1. Les nouvelles tables sont créés
2. Les données sont importées
3. Les indexes sont construits
4. Les triggers sont importés
5. Les contraintes d'integrité sont activées pour les nouvelles tables

Sauvegarde / Restauration RMAN

1. Utiliser l'outil Recovery Manager (RMAN) fourni par Oracle :
   - C'est la méthode recommandée
   - RMAN peut être utilisé en ligne de commande, Ou à travers une interface graphique dans le Database Control (OEM).

2. Procéder "à la main" avec des commandes du système d'exploitation et des scripts SQL

Types de sauvegarde (1)

1. Sauvegarde base fermée
   - Une base de données restaurée à partir d'une sauvegarde base fermée peut être ouverte immédiatement : il est inutile d'appliquer les fichiers de journalisation.
2. Sauvegarde base ouverte
   - Lorsqu'une base de données est restaurée à partir d'une sauvegarde incohérente, il faut appliquer les fichiers de journalisation pour rendre les fichiers cohérents.
   - Possible lorsque la base de données fonctionne en mode ARCHIVELOG.

Types de sauvegarde (2)

1. Sauvegarde complète
   - Sauvegarde de la totalité de la base de données.
2. Sauvegarde partielle
   - Sauvegarde incluant uniquement une partie de la base.
   - Elles sont incohérentes entre elles. Pour qu'elles soient exploitables en restauration, il faut que la base de données fonctionne en mode ARCHIVELOG.
3. Sauvegarde incrémentale
   - Sauvegarde qui ne contient que les blocs modifiés depuis la dernière sauvegarde ;
   - Peut être complète ou partielle.

Questions

Le mode ARCHIVELOG est-il nécessaire dans ces cas :

1 - La base ne peut pas être fermée pour être sauvegardée

2 - Les pertes de données ne sont pas acceptables

3 - Les pertes de données sont tolérées

Sauvegardes incrémentales

Sauvegarde incrémentale différentielle de niveau n

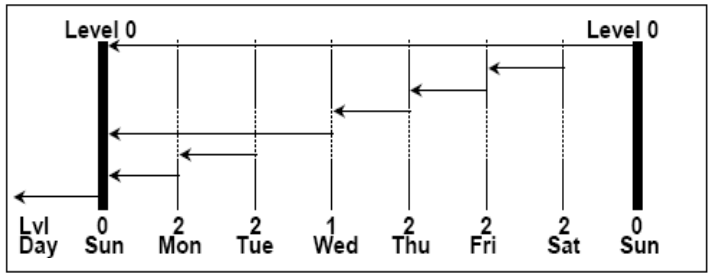
- Elle sauvegarde tous les blocs modifiés depuis la dernière sauvegarde de niveau inférieur ou égale à n

Sauvegarde incrémentale cumulative de niveau n

- Elle sauvegarde tous les blocs modifiés depuis la dernière sauvegarde incrémentale de niveau 0 (sauvegarde complète)

Stratégie de sauvegarde incrémentale différentielle (Exemple)

- Durée de sauvegardes de la base de données complète : 4h
- La base de données est en ligne 24h/24, 7J/7
- Baisse d'activités le Mercredi



Activer/désactiver le mode ARCHIVELOG

1. Il faut tout d'abord se connecter à la base en administrateur :
   - SQLPLUS /NOLOG
   - CONNECT / AS SYSDBA
2. Arrêter la base : SHUTDOWN
3. Démarrer la base en mode mount : STARTUP MOUNT
4. Modifier le mode d'archivage : ALTER DATABASE ARCHIVELOG (ou NOARCHIVELOG)
5. Ouvrir la base : ALTER DATABASE OPEN
6. Vérifier que tout c'est bien passé : ARCHIVE LOG LIST