Université des sciences et de la Technologie
Houari Boumediene
USTHB – Alger

Département d'Informatique

ADMINISTRATION ET TUNING DE BASES DE DONNÉES

OPTIMISATION DES REQUÊTES

Responsable

Dr K. Boukhalfa

ÉVALUATION DES REQUÊTES: SURVOL

❑ **Évaluation d'une requête SQL:**

- Analysée syntaxiquement, ensuite transformée en plan d'évaluation.
- **Plan d'évaluation**: Arbre d'opérations de l'algèbre relationnelle avec un choix d'algorithme pour chaque opération.

❑ **Deux problématiques importantes dans l'optimisation:**

- Énumération des plans alternatifs pour une requête
- Estimation des coûts de ces plans et choix de celui estimé être le moins cher

❑ **On doit se montrer pragmatique:**

- **Idéalement**: Trouver le meilleur plan.
- **Pratiquement**: Éviter les pires plans!

STATISTIQUES ET CATALOGUES

L'évaluateur a besoin d'informations sur les relations ainsi que les indexes impliquées. Les *Catalogues* contiennent:

- pour chaque relation: # tuples (NTuples) et # pages (NPages)
- pour chaque index: # valeurs de clés distinctes (NKeys) et # pages (NPages)
- pour chaque index à arbre: Hauteur de l'Index (Height), plus petites / plus grande valeurs de clé (Low/High)

Catalogues rafraîchis périodiquement.

- Automatiquement, Manuellement

D'autres types d'infos détaillées (p.ex., histogrammes des valeurs dans certains champs) sont aussi stockées.

- **Histogramme:** structure donnant une approximation de la distribution des valeurs des données

CHEMINS D'ACCÈS ET CORRESPONDANCE D'INDEX

- **Un chemin d'accès est une méthode pour puiser les tuples:**
  - Utilisation du Scannage du fichier, ou d'un index qui correspond à (''match'') une sélection (WHERE) de la requête.
- **Un index à arbre *correspond à* une conjonction de termes qui mentionnent seulement des attributs d'un *préfixe* de la clé de recherche.**
  - P.ex., l'index à arbre sur *<a, b, c>* correspond à la sélection *a=5 AND b=3* et à *a=5 AND b>6*, mais pas à *b=3*.
- **Un index à hachage *correspond à* une conjonction de termes qui a un terme de la forme *attribut = valeur* pour chaque attribut de la clé de recherche de l'index.**
  - P.ex., l'index à hachage sur *<a, b, c>* correspond à *a=5 AND b=3 AND c=5*; mais pas à *b=3*, ni à *a=5 AND b=3*, ni à *a>5 AND b=3 AND c=5*.

TECHNIQUES D'OPTIMISATION

❑Expressions Algébriques

- ❑ Transformation des requêtes pour obtenir la meilleure séquence d'opérations
- ❑ RBO : Rules Based Optimisation

❑Opérations Algébriques

- ❑ Coût des différentes stratégies possibles en fonction des caractéristiques des fichiers sur lesquels sont implantées les relations
- ❑ CBO : Cost Based Optimisation

Généralement les SGBDs commerciaux combinent les deux stratégies

- ❑ Oracle utilise actuellement CBO seulement

5

RULES BASED OPTIMISATION

Requête (SQL)

↓ Traduction

Requête en algèbre

↓ Transformation en fonction des propriétés des opérateurs

Plan optimal

6

PROPRIÉTÉS DES OPÉRATEURS

❑ Commutativité pour jointure et produit

❑Associativité pour jointure et produit

❑ Il existe N!/2 arbres de jointure de N relations.

7

PROPRIÉTÉS DES OPÉRATIONS (SUITE)

❑Groupage des projections

- ❑ $\pi A1, \ldots, An\ (\pi B1, \ldots., Bm(E)) = \pi A1, \ldots, An\ (E)$ --- {A1, …, An} inclus dans {B1, …, Bm}

❑Groupage des sélection

- ❑ $\sigma_{F1}\ (\sigma_{F2}\ (E)) = \sigma_{F1 \wedge F2}(E)$

❑Exemple : $\sigma_{age>20}\ (\sigma_{genre='F'}(Étudiant)) = \sigma_{(age>20)\ \wedge\ (genre='F')}(Étudiant)$

❑Inversion σ et π

- ❑ $\pi_{A1, \ldots, An}(\sigma_F(E)) = \sigma_F(\pi_{A1, \ldots, An}(E))$ si F porte sur A1, …, An
- ❑ $\pi_{A1, \ldots, An}(\sigma_F(E)) = \pi_{A1, \ldots, An}\ (\sigma_F(\pi_{A1, \ldots, An, B1, \ldots, Bn})\ (E))$ si F porte sur B1, …, Bm qui ne sont pas parmi A1, …, An

8

PROPRIÉTÉS DES OPÉRATIONS (SUITE)

Inversion Sélection-Produit

- $\sigma_F (E1 * E2) = \sigma_F (E1) * E$ si F ne porte que sur les attributs de E1
- $\sigma_F (E1 * E2) = \sigma_{F1} (E1) * \sigma_{F2} (E2)$ Si F = F1 ∧ F2 et Fi ne porte que sur les attributs de Ei
- $\sigma_F (E1 * E2) = \sigma_{F2} (\sigma_{F1} (E1) * E2)$ si F1 porte sur les attributs de E1 et F2 sur ceux de E1 et E2
- Exemple : $\sigma_{Genre='F'}$(Étudiant * Cours) = $\sigma_{Genre='F'}$(Étudiant) * Cours

Inversion Sélection-Union

- $\sigma_F (E_1 \cup E_2) = \sigma_F(E_1) \cup \sigma_F(E_2)$

Inversion Sélection-Diffétrence

- $\sigma_F (E_1 - E_2) = \sigma_F(E_1) - \sigma_F(E_2)$

Inversion Projection-Produit

Inversion Projection-Union

9

PRINCIPES D'OPTIMISATION DES EXPRESSIONS ALGÉBRIQUES

1. **Exécuter les sélections aussitôt que possible**
2. **Réduire la taille des relations à manipuler**
   - **Combiner les sélections avec un produit cartésien pour aboutir à une jointure**
3. **Combiner des séquences d'opérations unaires (σ,Π)**
4. **Mémoriser les sous-expression commune**

10

EXÉCUTER LES SÉLECTIONS AUSSITÔT QUE POSSIBLE

Appliquer ces règles (remplacer membre gauche par droit)

1. $\sigma_F (E1 \cup E2) = \sigma_F(E1) \cup \sigma_F(E2)$
2. $\sigma_F (E1 - E2) = \sigma_F(E1) - \sigma_F(E2)$
3. $\sigma_F (E1 * E2) = \sigma_F (E1) * E2$ si F ne porte que sur les attributs de E
4. $\sigma_F (E1 * E2) = \sigma_{F1} (E1) * \sigma_{F2} (E2)$ si F = F1 ∧ F2 et Fi ne porte que sur les attributs de Ei
5. $\sigma_F (E1 * E2) = \sigma_{F2} (\sigma_{F1} (E1) * E2)$ si F porte sur les attributs de E1 et F2 sur ceux de E1 et E2

11

COMBINER LES SÉLECTION AVEC UN PRODUIT CARTÉSIEN POUR ABOUTIR À UNE JOINTURE

❑ **Transformation si possible du produit cartésien en jointure:**

- Si R * S est l'argument d 'une sélection σ_F alors si F est une comparaison entre attributs de R et de S, transformer R * S en jointure : $\sigma_F (R * S) = (R \infty_F) S$

❑**Attention : si F est une expression ne portant que sur les attributs de R:**

- $\sigma_F (R * S) = \sigma_F (R) * S$

12

SOUS-EXPRESSIONS COMMUNES À UNE EXPRESSION

- Si une expression contient plusieurs fois la même sous-expression
- Si cette sous-expression peut être lue en moins de temps qu'il faut pour la calculer
- Matérialiser la sous-expression commune
- **Principe de vues matérialisées**

13

UN ALGORITHME D'OPTIMISATION

1. Séparer chaque sélection $\sigma_{F1 \wedge \ldots \wedge Fn}(E)$ en une cascade $\sigma_{F1}(\sigma_{F1} \ldots (\sigma_{Fn}(E)))$
2. Descendre chaque sélection le plus bas possible dans l'arbre algébrique
3. Descendre chaque projection aussi bas possible dans l'arbre algébrique
4. Combiner des cascades de sélection et de projection dans une sélection seule, une projection seule, ou une sélection suivie par une projection
5. Regrouper les nœuds intérieurs de l'arbre autour des opérateurs binaires (*, -, U). Chaque opérateur binaire crée un groupe
6. Produire un programme comportant une étape pour chaque groupe, à évaluer dans n'importe quel ordre mais tant qu'aucun groupe ne soit évalué avant ses groupes descendants.

14

QUELQUES PROBLÈMES

- La restructuration d'un arbre sous-entend souvent la permutation des opérateurs binaires
  - Cette optimisation ignore la taille de chaque table, les facteurs de sélectivité, etc.
  - Aucune estimation de résultats intermédiaires
- Exemple:

(Ouvrier ∞ Usine) ∞ Contrat

Si cardinalité (Usine) <<<Cardinalité(Contrat) << Cardinalité(Ouvrier) alors l'expression suivante a des chances d'être plus performante:

(Usine ∞ Contrat) ∞ Ouvrier

→ Optimisation basée sur les modèles de coût

15

MÉTHODES D'OPTIMISATION BASÉES SUR LES MODÈLES DE COÛTS

16

NOTIONS UTILES

- Les tables relationnelles sont stockées physiquement dans des fichiers sur disque
- Lorsqu'on veut accéder aux données, on doit transférer le contenu pertinent des fichiers dans la mémoire
  - Fichier = séquence de tuples
  - Bloc (page) = unité de transfert de données entre disque et mémoire
  - Facteur de blocage = nombre de tuples d'une relation qui «tiennent» dans un bloc
  - Coût de lecture (ou écriture) d'une relation = nombre de blocs à lire du disque vers la mémoire (ou à écrire de la mémoire vers le disque)

17

ESTIMATION DU COÛT DES OPÉRATIONS PHYSIQUES

- **TempsES : temps d'accès à mémoire secondaire (MS)**
- **TempsUCT : Souvent négligeable**
- **TailleMC : espace mémoire centrale**
- **TailleMS : espace mémoire secondaire**

18

MODÈLE DU COÛT D'UNE ENTRÉE- SORTIE EN MÉMOIRE SECONDAIRE

Paramètre	Signification
TempsESDisque(n)	Temps total de transfert (lecture ou écriture) de n octets du disque
TempsTrans(n)	Temps de transfert des n octets sans repositionnement
TempsPosDébut	Temps de positionnement au premier octet à transférer (ex : 10 ms)
TempsRotation	Délai de rotation (ex : 4 ms)
TempsDépBras	Temps de déplacement du bras (ex : 6 ms)
TauxTransVrac	Taux de transfert en vrac (ex : 2MB/sec)
TempsESBloc	Temps de transfert d'un bloc (ex : 11 ms)
TempsTrans	Temps de transfert d'un bloc sans repositionnement (ex : 1 ms)
TailleBloc	Taille d'un bloc (ex : 2K octets)

19

STATISTIQUES AU SUJET DES TABLES

Statistique	Signification
||T||	Nombre de lignes de la table T
TailleLigne	La taille d'une ligne de la table T
FBT	Facteur de blocage moyen de T
NDIST	Nombre de valeurs distinctes de la colonne pour la table T
Min(colonne)	Valeur minimum de la colonne de T
Max(colonne)	Valeur maximum de la colonne de T
|T|	Nombre de pages de la table T

20

FACTEUR DE SÉLECTIVITÉ

- ||(σ(R))|| = SF * ||R||
  - SF (A = valeur) = 1 / NDIST(A)
  - SF(A > valeur) = (max(A) - valeur) / (max(A) - min(A))
  - SF(A < valeur) = (valeur - min(A)) / (max(A) - min(A))
  - SF(A IN liste valeurs) = (1/NDIST(A)) * CARD(liste valeurs)
  - SF(P et Q) = SF(P) * SF(Q)
  - SF(P ou Q) = SF(P) + SF(Q) - SF(P) * SF(Q)
  - SF(not P) = 1 - SF(P)

Le coût dépend de l'algorithme (index, hachage ou balayage).

21

FACTEUR DE SÉLECTIVITÉ

La taille d'une jointure est estimée par la formule suivante:

||R1 ∞ R2|| = Sel * ||R1|| * ||R2||;

avec sel: la sélectivité de la jointure

22

EXEMPLES

```
SELECT *
FROM R1, R2
```

SF =1

```
SELECT *
FROM R1
WHERE A = valeur
```

SF = 1/NDIST(A) avec un modèle uniforme

23

ALGORITHMES GÉNÉRAUX POUR LES OPÉRATEURS RELATIONNELS

24

LA SÉLECTION

❑ **Sélection sans index**

SELECT NSS, Nom FROM Ouvrier WHERE Salaire > 15000

❑ **Plan d'exécution**

- pour chaque tuple t de ouvrier faire -- accès séquentiel
- si **t.salaire > 15000** alors **réponse = réponse** ∪ πNSS, Nom (Ouvrier)

25

ALGORITHMES DE JOINTURE

❑ **Jointure sans index**

- ❑ **Jointure par boucles imbriquées (nested loop)**
- ❑ **Jointure par tri-fusion (sort-join)**
- ❑ **Jointure par hachage (hash-join)**

❑ **Jointure avec index**

- ❑ **Jointure avec boucles indexées**

26

JOINTURE PAR BOUCLES IMBRIQUÉES (JBI)

❑ **R ∞_F S**

❑ **Principe:**

- ❑ La première table est lue séquentiellement et de préférence stockée entièrement en mémoire
- ❑ Pour chaque tuple de R, il y a une comparaison avec les tuples de S

27

JOINTURE PAR BOUCLES IMBRIQUÉES

```
POUR chaque ligne lR de R
        POUR chaque ligne lS de S
                SI θ sur lR et lS est satisfait
                        Produire la ligne concaténée à partir de lR et lS
                FINSI
        FINPOUR
FINPOUR
```

❑ + Algorithme simple

❑ Complexité: (||R|| + (||R||*||S||)

❑ Si la table extérieure tient en mémoire: une seule lecture des deux tables suffit.

28

JOINTURE PAR TRI FUSION

- Plus efficace que les boucles imbriquées pour les grosses tables
- Principe:
  - Trier les deux tables sur les colonnes de jointure
  - Effectuer la fusion
  - Complexité: Le tri qui coûte cher

29

JOINTURE PAR TRI FUSION

Trier R et S par tri externe et réécrire dans des fichiers temporaires
Lire groupe de lignes GR(cR) de R pour la première valeur cR de clé de jointure
Lire groupe de lignes GS(cS) de S pour la première valeur cS de clé de jointure
TANT QUE il reste des lignes de R et S à traiter

```
SI cR = cS
        Produire les lignes concaténées pour chacune des combinaisons de lignes de
                GR(cR) et GS(cS);
        Lire les groupes suivants GR(cR) de R et GS(cS) de S;
 SINON
            SI cR < cS
                    Lire le groupe suivant GR(cR) de R
              SINON
                  SI cR > cS
                          Lire le groupe GS(cS) suivant dans S
                  FINSI
            FINSI
 FINSI
FIN TANT QUE
```

30

JOINTURE PAR TRI FUSION

- Très efficace quand une des deux tables est petite (1, 2, 3 fois la taille de la mémoire)
- Algorithme en option dans Oracle

31

PRINCIPE DE LA JOINTURE PAR HACHAGE

1. Hacher la plus petite des deux tables en n fragments
2. Hacher la seconde table, avec la même fonction, en n autres fragments
3. Réunir fragments par paire, et faire la jointure entre les fragments

32

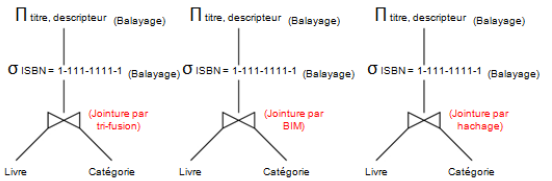
COMMENT SE FAIT L'OPTIMISATION

- **Chercher le meilleur plan d 'exécution?**
  - coût excessif
- **Solution approchée à un coût raisonnable**
  - Générer les alternatives
    - heuristiques
- **Choisir la meilleure estimation approximative du coût**

33

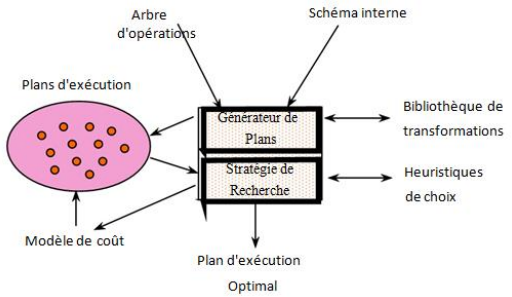
PLUSIEURS PLANS D'EXÉCUTION POUR UN ARBRE ALGÉBRIQUE

- **Pour chaque opération logique**
  - plusieurs choix d 'opérations physiques



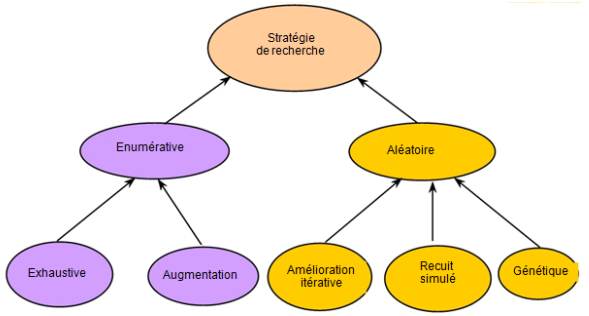
34

CHOIX DU MEILLEUR PLAN



35

DIFFÉRENTES STRATÉGIES



36

OPTIMISATION BASÉE SUR MODÈLE DE COÛT

- Soit Q une requête à optimiser
- Procédure

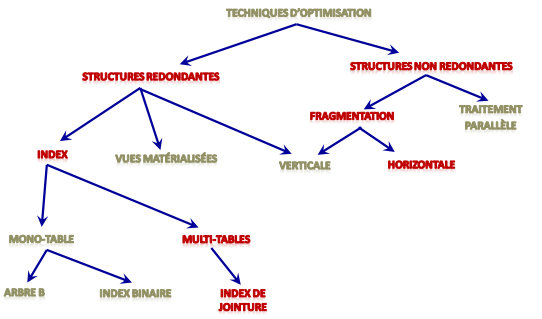
1. Énumérer tous les plans $\{P_1, \ldots, P_m\}$ pour chaque requête (notons que chaque requête possède un ensemble d'opérations $O_1, \ldots, O_k$)
2. Pour chaque plan P_j
   - Pour chaque opération O_i du plan P_j, énumérer les chemins d'accès
   - Sélectionner le chemin ayant le coût le moins élevé $\quad Coût(P_i) = \Sigma_{(i=1,\,k)} \min(O_i)$
3. Coût (Q): $\Sigma_{(h=1,\,m)}$ Coût(P_h)

STRUCTURES D'OPTIMISATION

TECHNIQUES UTILISÉES

- **Vues matérialisées**
  - Une vue est une requête nommée
  - Une vue matérialisée : les données résultant de sa requête sont stockées et maintenues
- **Index**
  - Structures permettant d'associer à une clé d'un n-uplet l'adresse relative de cet n-uplet
- **Traitement parallèle**
- **Groupement**
- **Fragmentation**

CLASSIFICATION



VUES MATÉRIALISÉES

- **Exigence de ressources:**
  - Espace disque
  - Coût de maintenance (rafraîchissement des données)
  - Coût de calcul

➡ Impossibilité de matérialiser toutes les vues

- **Problème de Sélection des Vues (PSV):**

- Sélectionner un ensemble de vues afin de maximiser ou minimiser une fonction objectif sous une contrainte

- Fonction « objectif »:

Optimiser le coût d'exécution des requêtes ou Optimiser le coût de maintenance

41

EXEMPLE

```
CREATE MATERIALIZED VIEW VUE1
ENABLE QUERY REWRITE
AS
SELECT *
FROM Vente, Client, Article
 WHERE Vente.noClient = Client.noClient AND
Vente.noArticle = Article.noArticle
```

42

MAINTENANCE DES VUES MATÉRIALISÉES

- Vues matérialisées sont calculées à partir des sources (tables)

- Mise à jour des sources implique la mise à jour des vues

- Deux méthodes de maintenance
  - Statique
  - Incrémentale

43

RÉÉCRITURE DES REQUÊTES

- Processus de réécriture:
  - Après le processus de sélection des vues, toutes les requêtes définies sur l'entrepôt doivent être réécrites en fonction des vues

- Sélectionner la meilleure réécriture pour une requête est une tâche difficile.

- Processus supporté dans la plupart des SGBD multidimensionnels (Oracle)

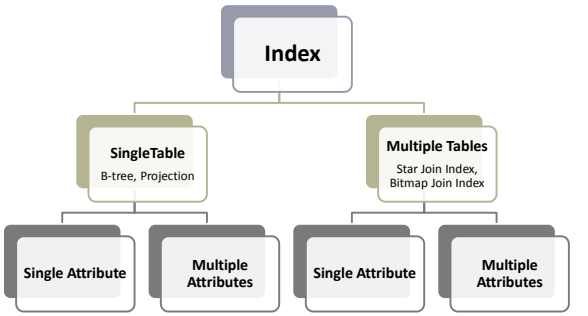
44

INDEX AVANCÉS

- Index populaires dans les entrepôts:
  - Index simple: sur une seule table ou une seule vue (arbre B, index binaire, index de projection)
  - Index de jointure: sur plusieurs tables dans un schéma en étoile
    - index de jointure en étoile
    - Index de jointure binaire
- Problème de sélection d'index
- Algorithmes de sélection d'index

45

CLASSIFICATION DES INDEX



46

TECHNIQUES D'INDEXATION

- Index bitmap (index binaire)
  - Populaire dans les produits OLAP
    - Microsoft SQL server, Sysbase IQ, Oracle
    - Un vecteur de bits pour chaque valeur d'attribut
  - Opération bit à bit pour l'exécution de requêtes
    - Comparaison
    - Jointure
    - Agrégation
    - Plus compact que les B+ arbres (compression)

47

PRINCIPE DE L'INDEX BITMAP

Soit un attribut A, ayant prenant n valeurs possibles $[v_1, ..., v_n]$ (domaine)

Création d'un index bitmap sur l'attribut A:

- On crée n tableaux de bit, un pour chaque valeur v_i
- Le tableau contient un bit pour chaque tuple t
- Le bit d'un tuple t est à 1 si: $t.A = v_i$, à 0 sinon

Index binaire sur l'attribut Fonction

ROWID(RID)	Soudeur	Fraiseur	Sableur	Tourneur
00055 :000 :0023	0	1	0	0
00234 :020 :8922	1	0	0	0
19000 :328 :6200	0	0	0	1
21088 :120 :1002	0	0	1	0

Table Employé

ROWID(RID)	N°E	Nom	Fonction
00055 :000 :0023	1	Karim	Fraiseur
00234 :020 :8922	2	Hichem	Soudeur
19000 :328 :6200	3	Salim	Tourneur
21088 :120 :1002	4	Ali	Sableur

48

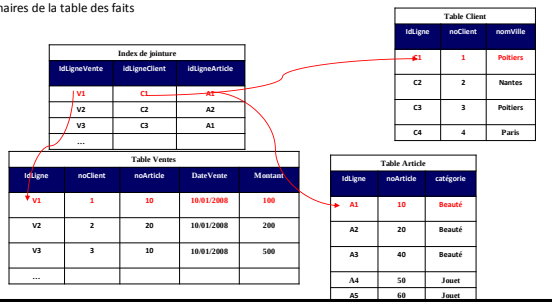
INDEX DE JOINTURE

Pré-calculent une opération de jointure

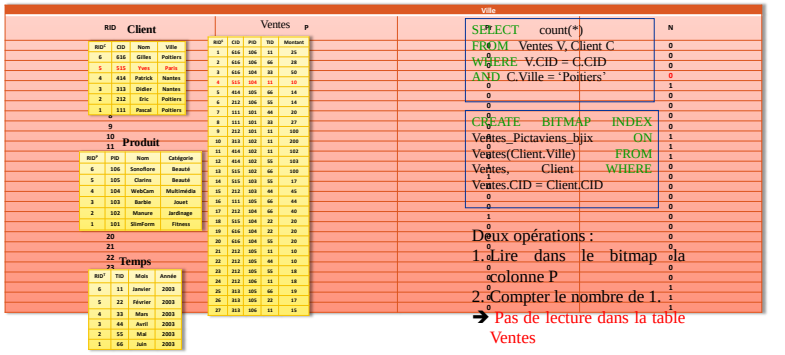
Utilisés avec les schémas en étoile

Maintient des relations entre

- Une clé étrangère (identifiant des tables de dimensions)
- Les clés primaires de la table des faits



INDEX DE JOINTURE BITMAP : EXEMPLE




```
SELECT   count(*)
FROM   Ventes V, Client C
WHERE  V.CID = C.CID
AND  C.Ville = 'Poitiers'
```

```
CREATE   BITMAP   INDEX
Ventes_Pictaviens_bjix   ON
Ventes(Client.Ville)   FROM
Ventes,   Client   WHERE
Ventes.CID = Client.CID
```

Deux opérations :

1. Lire dans le bitmap la colonne P
2. Compter le nombre de 1.

➔ Pas de lecture dans la table Ventes

CARACTÉRISTIQUES DES BITMAPS

- Les opérations de comparaison, jointure et d'agrégation sont réduites à une arithmétique sur les bits, d'où un traitement plus efficace
- La réponse à une requête multidimensionnelle va consister à faire l'intersection des vecteurs de bits des diverses dimensions
- Réduction significative en espace et nombre d'accès en mémoire secondaire

INTÉRÊT DES INDEX DE JOINTURE BINAIRE

```
CREATE BITMAP INDEX EMPDEPT_IDX ON
 EMP1(DEPT1.DEPTNO)
FROM EMP1, DEPT1
WHERE EMP1.DEPTNO = DEPT1.DEPTNO
```

```
SELECT /*+ INDEX(EMP1 EMPDEPT_IDX) */ COUNT(*)
FROM EMP1, DEPT1
WHERE EMP1.DEPTNO = DEPT1.DEPTNO;
```

```
COUNT(*)
----------
        14
Elapsed: 00:00:00.67
```

BITMAP JOIN INDEX EXAMPLE 2

Deux tables EMP5/EMP6 ayant 2 million d'instances chacune, empno est la clé étrangère

```
create bitmap index emp5_j6 on
emp6(emp5.empno) from emp5, emp6
where emp5.empno=emp6.empno;
```

Index created.

Elapsed: 00:02:29.91

53

BITMAP JOIN INDEX EXAMPLE 2

```
select count(*)
from emp5, emp6
where emp5.empno=emp6.empno
```

```
COUNT(*)
----------
   2005007
```

Elapsed: 00:01:07.18

54

BITMAP JOIN INDEX EXAMPLE 2

Plan d'exécution

```
0      SELECT STATEMENT Optimizer=CHOOSE
1    0   SORT (AGGREGATE)
2    1     NESTED LOOPS
3    2       TABLE ACCESS (FULL) OF 'EMP6'
4    2       INDEX (RANGE SCAN) OF 'EMP5I_EMPNO' (NON-UNIQUE)
```

55

BITMAP JOIN INDEX EXAMPLE 2

FORCER L'UTILISATION DE L'INDEX

```
select /*+ index(emp6 emp5_j6) */ count(*)
from emp5, emp6
where emp5.empno=emp6.empno
```

```
COUNT(*)
----------
   2005007
```

Elapsed: 00:00:00.87! Comme avec les tables de petites tailles!

56

BITMAP JOIN INDEX - 10,000 PLUS RAPIDE

Plan d'exécution

```
0      SELECT STATEMENT Optimizer=CHOOSE
1   0  SORT (AGGREGATE)
2   1  BITMAP CONVERSION (COUNT)
3   2  BITMAP INDEX (FULL SCAN) OF 'EMP5_J6'
```

57

SELECTION OF BJI : FORMALIZATION

INPUT

- Set of dimension tables D={D_1, D_2, …, D_d}
- Fact Table F
- Workload of frequently queries Q
- **Storage Space S**

OUTPUT

- **Set S_BJI of BJI**

OBJECTIVES

- Reduce execution cost of Q
- Size (S_BJI) ≤ S

58

SELECTION OF BJI : COMPLEXITY

- A={A_1, A_2, …, A_K} is a set of indexable attributes
- A potential BJI is defined on a subset of A
- N the number of potential BJI

Select One BJI

$$N = \binom{k}{1}+\binom{k}{2}+\ldots\ldots+\binom{k}{k} = 2^k - 1$$

Select More Than One BJI

$$N = \binom{2^k-1}{1}+\binom{2^k-1}{2}+\ldots\ldots+\binom{2^k-1}{2^k-1} = 2^{2^k-1}$$

59

Fragmentation de Données (Partitionnement)

60

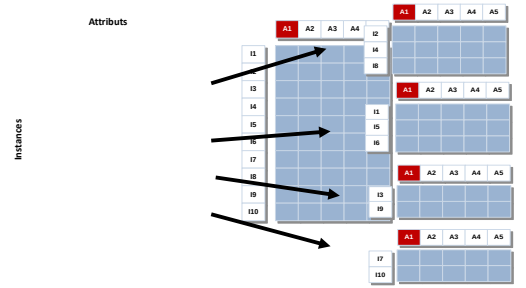
LA FRAGMENTATION DE DONNÉES

- Définition
  - ❑ Décomposer les objets de la BD (relation, index, vues) en un ensemble de morceaux appelés Partitions.
- Fragmentation horizontale
  - ❑ La table est fragmentée par rapport à ses instances en un ensemble de lignes.
- Fragmentation verticale
  - ❑ La table est fragmentée selon ses attributs en un ensemble de colonnes.
- Fragmentation mixte
  - ❑ La table est fragmentée horizontalement et verticalement.

61

TYPES DE FRAGMENTATION

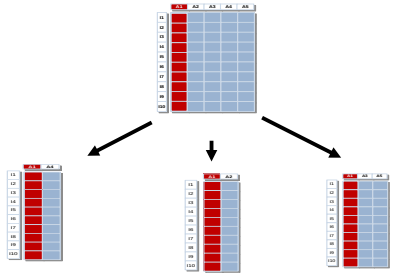
- Fragmentation horizontale
  - Décomposer les objets en un ensemble de lignes (instances)



62

TYPES DE FRAGMENTATION

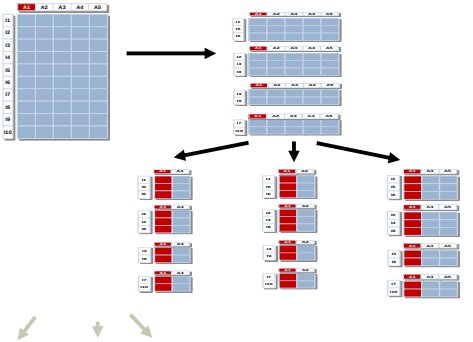
- Fragmentation verticale
  - ❑ Décomposer les objets en un ensemble de colonnes (attributs).



63

TYPES DE FRAGMENTATION

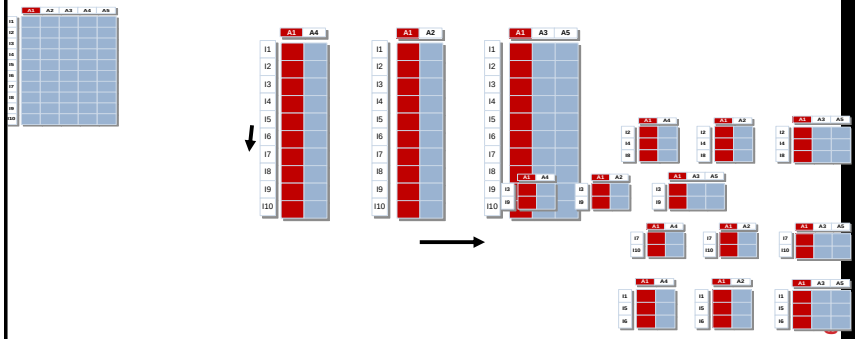
- Fragmentation mixte
  - ❑ Horizontale suivie d'une verticale.



64

TYPES DE FRAGMENTATION

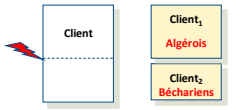
- **Fragmentation mixte**
  - ❑ Verticale suivie d'une horizontale



FRAGMENTATION HORIZONTALE PRIMAIRE ET DÉRIVÉE (I)

- **Fragmentation horizontale primaire (FHP)**
  - ❑ Fragmenter une table en utilisant les prédicats de sélection définis sur cette table

    Prédicat : Attribut θ Valeur, θ ∈{=,<, ≤, >, ≥,} et valeur ∈Domaine(Attribut).
  - ❑ Exemple: Client (Client_id, Nom, Ville)
    - $Client_1 : \sigma_{Ville='Alger'}(Client)$
    - $Client_2 : \sigma_{Ville='Béchar'}(Client)$
- **Impact de la FHP sur les requêtes**
  - ❑ **Optimisation des sélections**




```
Select Nom, Age
From Client
Where Ville=Béchar
```

Client

Client1 Algérois

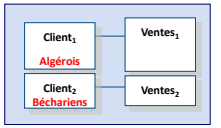
Client2 Béchariens

```
Select Nom, Age
From Client2
```

66

FRAGMENTATION HORIZONTALE PRIMAIRE ET DÉRIVÉE (II)

- **Fragmentation horizontale dérivée (FHD)**
  - ❑ Fragmenter une table (S) selon des attributs d'une autre table (T) : (existence de lien entre S et T)
  - Ventes(Client_id, Produit_id, Date, Montant)
    - $Ventes_1$=Ventes ⇔$Client_1$
    - $Ventes_2$=Ventes ⇔ $Client_2$



- **Impact de la FHD sur les requêtes**
  - ❑ **Optimisation de la jointure entre S et T**

```
Select sum(Montant)
From Client C, Ventes V
Where C.CID=V.CID and
Ville=Béchar
```

Ventes

Client

Ventes1 Ventes2

Client1 Algérois

Client2 Béchariens

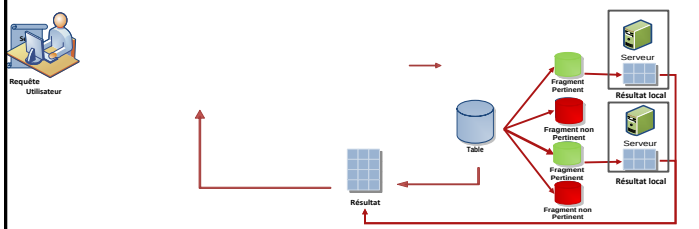
```
Select sum(Montant)
From Ventes2
```

67

INTÉRÊT DE LA FRAGMENTATION HORIZONTALE(I)

- **Améliorer la performance**

1. **Partition Elimination (Pruning)**
   - Elimination des partitions non pertinentes
   - Possibilité d'exécution parallèle des sous requêtes

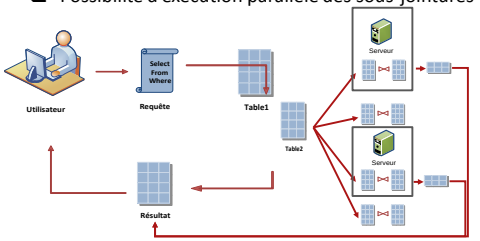


68

INTÉRÊT DE LA FRAGMENTATION HORIZONTALE(II)

- Amélioration de la performance

2. Partition Wise Joins
   - ❑ Elimination des jointures non pertinentes
   - ❑ Possibilité d'exécution parallèle des sous-jointures



69

INTÉRÊT DE LA FRAGMENTATION HORIZONTALE(III)

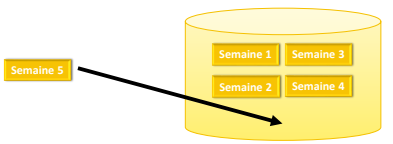
- Améliorer la manageabilité
  - Fragmentation horizontale préserve le schéma logique
    - ❑ Toutes les opérations se font au niveau partition
    - ❑ Possibilité de manipulation individuelle ou collective des partitions
  - Manipuler une partition à la fois.
    - ❑ Possibilité de définir des index locaux aux partitions
    - ❑ Existences de plusieurs Fonctions de manipulation des partitions

Fonction	Signification
ADD PARTITION	Ajouter une partition à une table déjà fragmentée
COLESCE PARTITION	Redistribuer les tuples d'une partition dans les autres partitions
DROP PARTITION	Supprimer une partition ainsi que son contenu
EXCHANGE PARTITION	Convertir une table non fragmentée en une partition d'une autre table et inversement
MERGE PARTITION	Fusionner deux partitions dans une seule
SPLIT PARTITION	Eclater une partition en deux partitions
TRUNCATE PARTITION	Vider une partition sans la supprimer

70

EXEMPLE D'AJOUT DE DONNÉES

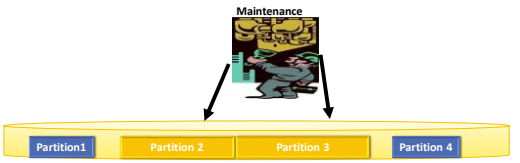
- Alimentation hebdomadaire d'une table
  - ❑ L'administrateur partitionne sa table par semaine
  - ❑ L'alimentation de l'entrepôt consiste alors à ajouter une partition à la table
  - ❑ Aucune autre partition ne sera touchée.



71

INTÉRÊT DE LA FRAGMENTATION HORIZONTALE(IV)

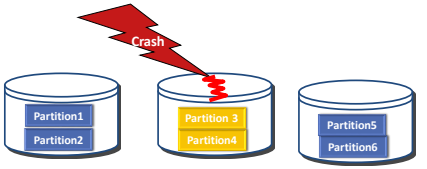
- Améliorer la maintenance et la disponibilité
  - La manipulation au niveau partition permet
    - ❑ Cibler la maintenance sur certaines partitions
    - ❑ Minimiser le temps de maintenance
  - La possibilité de stocker les partitions sur des emplacements différents permet
    - ❑ L'effet des pannes de disques est limité
    - ❑ Partitions saines toujours disponibles



72

AMÉLIORER LA MAINTENANCE ET LA DISPONIBILITÉ

- Améliorer la maintenance et la disponibilité
  - La possibilité de stocker les partitions sur des emplacements différents permet
    - L'effet des pannes de disques est limité
    - Partitions saines toujours disponibles



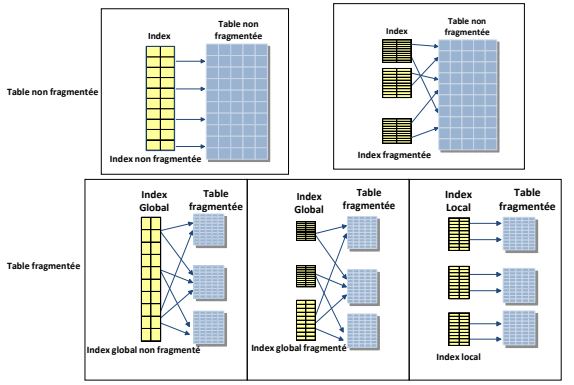
73

FRAGMENTATION HORIZONTALE ET INDEX

- **Un index peut être défini sur une table fragmentée ou non**
- **Un index défini sur une table fragmentée peut être fragmenté ou non.**
- **Index global**
  - Peut être non fragmenté défini sur une table fragmentée
  - Peut être fragmenté où chaque partition de l'index référence plusieurs partitions de la table
- **Index local**
  - Equi-partitionné avec la table qu'il référence
  - Chaque partition de l'index référence une et une seule partition de la table

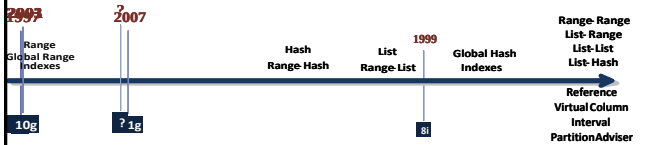
74

FRAGMENTATION ET INDEX



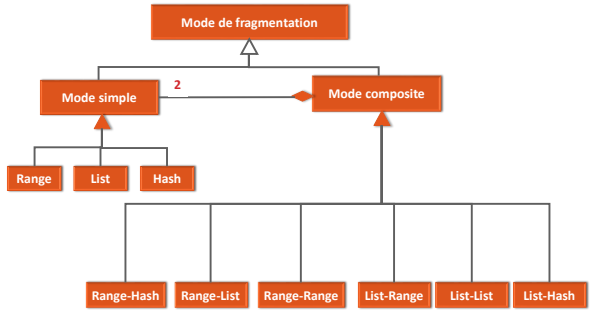
75

EVOLUTION INDUSTRIELLE



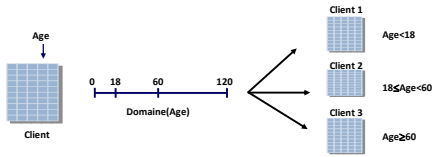
76

MODES DE FRAGMENTATION



77

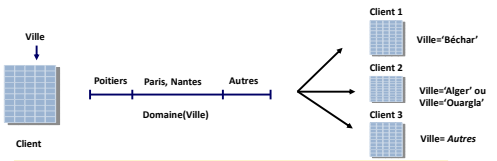
FRAGMENTATION PAR INTERVALLE « RANGE »




```
CREATE TABLE Client
(Client_id  NUMBER(5),
Nom  Varchar2(20),
Ville Varchar2(20),
Age Number(3),
Genre Varchar2(1))
PARTITION BY RANGE (Age)
(
PARTITION Client_Moins_18 VALUES LESS THAN (18) TABLESPACE TBSMoins27,
PARTITION Client_18_59 VALUES LESS THAN (60) TABLESPACE TBS27-59,
PARTITION Client_60_Et_Plus VALUES LESS THAN (MAXVALUES) TABLESPACE TBSPlus60
);
```

78

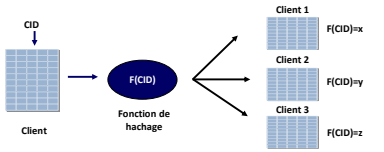
FRAGMENTATION PAR LISTE « LIST »




```
CREATE TABLE Client
(Client_id  NUMBER(5),
Nom  Varchar2(20),
Ville Varchar2(20),
Age Number(3),
Genre Varchar2(1))
PARTITION BY List (Ville)
(
PARTITION Client_Béchar VALUES ('Béchar') TABLESPACE TBSBECHAR,
PARTITION Client_Alg_Oua VALUES ('Alger','Ouargla') TABLESPACE TBSALGOUA,
PARTITION Client_Autres VALUES (DEFAULT) TABLESPACE TBSAUTRES
);
```

79

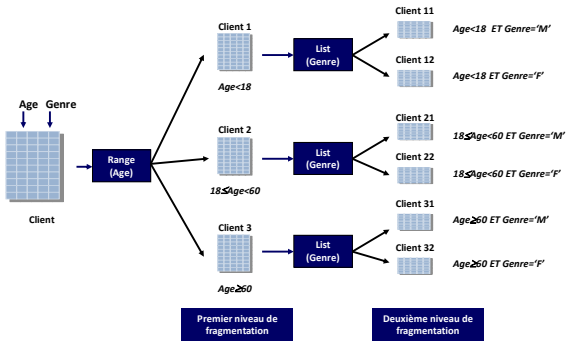
FRAGMENTATION PAR HACHAGE « HASH »




```
CREATE TABLE Client
(Client_id  NUMBER(5),
Nom  Varchar2(20),
Ville Varchar2(20),
Age Number(3),
Genre Varchar2(1))
PARTITION BY Hash (CID)
(
PARTITIONS 3
STORE IN (TBS1, TBS2, TBS3)
);
```

80

MODES DE FRAGMENTATION COMPOSITE



81

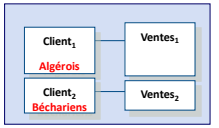
MODES DE FRAGMENTATION COMPOSITE

```
CREATE TABLE Client
(Client_id  NUMBER(5),
Nom  Varchar2(20),
Ville Varchar2(20),
Age Number(3),
Genre Varchar2(1))
PARTITION BY RANGE (Age)
SUBPARTITION BY LIST (Genre)
SUBPARTITION TEMPLATE(
SUBPARTITION  Client1 VALUES ('M') TABLESPACE TBSMasculin ,
SUBPARTITION Client2 VALUES ('F') TABLESPACE TBSFéminin )
(
PARTITION Client_Moins_18  VALUES LESS THAN (18),
PARTITION Client_18_59 VALUES LESS THAN (60),
PARTITION Client_60_Et_Plus  VALUES LESS THAN (MAXVALUES)
);
```

82

FRAGMENTATION DÉRIVÉE PAR LE MODE RÉFÉRENCE

- Fragmenter une table selon le schéma de fragmentation d'une autre table en utilisant le lien par clé étrangère.




```
CREATE TABLE Ventes
(Client_id  NUMBER(5),
Time_id  NUMBER(5)),
Montant NUMBER(20),
 CONSTRAINT order_items_fk
 FOREIGN KEY(Client_id) REFERENCES Client(Client_id)
 )
PARTITION BY REFERENCE(order_items_fk);
```

83