Systèmes d'information décisionnels (Data Warehouse)

Dr Dendani-Hadiby Nadjette
Université Badji Mokhtar Annaba
Département d'Informatique
n_dendani@yahoo.fr

2021/2022

Matérialisation des vues

La matérialisation des vues 1

- Une vue peut être désignée par l'expression de la requête qui la définit, une vue matérialisée est une table contenant le résultat d'une requête. La vue matérialisée a donc une existence physique. La matérialisation des vues améliore l'exécution des requêtes en précalculant les opérations les plus coûteuses ou des ensembles d'enregistrements qui évitent l'accès à l'entrepôt de données. Donc certaines requêtes ne nécessitent l'accès qu'aux vues matérialisées.
- Dans le contexte OLTP, les vues sont utilisées pour satisfaire d'autres objectifs tels que la sécurité, la confidentialité, etc.…
- La matérialisation des vues permet d'améliorer les performances des requêtes, mais elle peut être aussi utilisée pour fournir des données dupliquées.
- Les deux grands problèmes de la matérialisation des vues sont
  1. le problème de la sélection des vues à matérialiser
  2. le problème de la maintenance des vues matérialisées

La sélection des vues à matérialiser

- Dans un entrepôt de données, il est possible de spécifier un ensemble de requêtes à privilégier. En fonction de ces requêtes, l'ensemble des vues à matérialiser sera défini.
- Selon le type du modèle de données utilisé (MOLAP ou ROLAP), le problème de la sélection des vues à matérialiser sera vu différemment.
  - Dans le cas du modèle du type MOLAP, le cube de données est considéré comme structure principale. Chaque cellule du cube est considérée comme vue potentielle. Dans ce cas une structure de treillis sera utilisée pour déterminer les vues à matérialiser.
  - Dans le cas du modèle ROLAP, chaque requête est représentée par un arbre algébrique, chaque noeud non feuille est considéré comme vue potentielle

La sélection des vues à matérialiser

La matérialisation des vues est une technique bien connue et employée pour accélérer le processus de prise de décision. Dans ce contexte, il y a trois possibilités:

- • Matérialiser physiquement toutes les vues : cette approche donne le meilleur temps de réponse à une requête. Cependant, enregistrer toutes les vues n'est pas une alternative faisable pour les grands cubes de donnes, l'espace consommé devient alors très grand.
- • Ne rien matérialiser : cette approche est la meilleure en espace consommé mais elle est la plus mauvaise en temps de réponse, car On est obligés d'aller à chaque fois aux données brutes pour répondre à une requête.
- • Matérialiser seulement une partie des vues : Dans un cube de données, la valeur de beaucoup de cellules est calculable de celles d'autres cellules. C'est à dire qu'on peut déduire un ensemble de vues à partir d'autres (une vue contient un ensemble de cellules). Au lieu de matérialiser toutes les vues on matérialise un sous ensemble de vues qui permettent de répondre à n'importe quelle requête.

La matérialisation partielle est la solution la plus faisable

La sélection des vues à matérialiser

Le problème de la sélection des vues à matérialiser peut être vu de la manière suivante. D'après H. Gupta

- Etant donnée une contrainte de ressource S (Capacité de stockage ou coût de maintenance), le problème de la sélection des vues à matérialiser consiste à sélectionner un ensemble de vues {V1,V2,…,Vn} minimisant une fonction objectif (coût de traitement des requêtes) et satisfaisant la contrainte (figure).
- Les méthodes exactes qui résolvent se problème tentent d'énumérer toutes les vues possibles, afin de sélectionner un ensemble de vues optimales. Leur nombre est en général très grand. Si d est le nombre de dimensions dans un schéma, alors le nombre d'agrégations possibles est égal à $n=2^d$ et la complexité est de $O(2^n)$.
- Il a été montré que ce problème est NP-difficile [Gupta 99].

L'Algorithme de Glouton

- L'algorithme glouton (Greedy Algorithm) utilise une structure du treillis pour représenter les dépendances entre les vues. Les solutions offertes par cet algorithme garantissent un bénéfice d'au moins 63% du bénéfice de l'optimal.

L'algorithme glouton (Greedy Algorithme)

- On suppose un treillis de vues associé à un cube de données, avec un coût de traitement associé à chaque vue (le coût de traitement est égal au coût d'espace occupé). Soient c (v) le coût de traitement de la vue v, k le nombre de vues à sélectionner et S l'ensemble qui va contenir les k vues sélectionnées plus la vue sommet qui doit toujours être matérialisée (initialement S = {vue sommet}).
- Apres avoir sélectionné un ensemble S de vues, le bénéfice de la vue v relatif à S noté B (v,S), est défini comme suit :

1. Pour chaque vue $w \leq v$, on définit la quantité Bw par :
   a. Soit u la vue qui a le plus petit coût dans S tel que $w \leq u$ (u est l'ancêtre de w qui a le plus petit coût de traitement dans S).
   b. Si c (v) < c (u), alors Bw = c (u) – c (v) Sinon Bw = 0.
2. On définit le bénéfice de la vue v relatif à S noté B (v,S) par: $B(v,S)=\Sigma_{v\leq=w}B_w$

L’algorithme glouton (Greedy Algorithme)

- On calcule le bénéfice d'une vue v en considérant comment cette vue peut améliorer le coût de traitement des vues. Pour chaque vue w que couvre v, on comparons le coût de w en utilisant v et en utilisant l'évaluation la moins chère à partir de S. Si la présence de v améliore, On retient v pour la matérialisation.
- L'algorithme glouton pour sélectionner k vues à matérialiser est décrit comme suit :

S = {vue sommet};

Pour i = 1 à k faire

DEBUT

Sélectionner la vue v non dans S qui maximise B(v,S);

S = S U {v};.

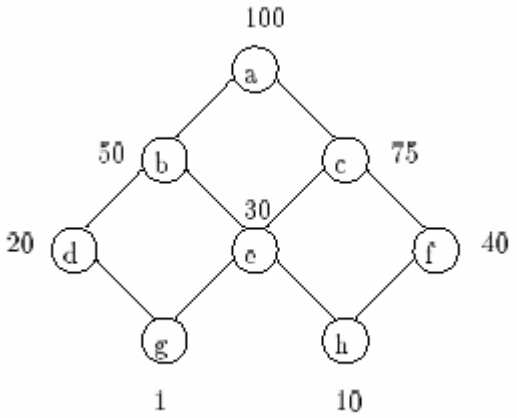
FIN ;

- S est l'ensemble des vues à matérialiser

Exemple (1)

- Lorsqu'on calcule les bénéfices, on commence avec la supposition que chaque vue est évaluée en utilisant la vue sommet a, car initialement S = {a}.

	Premier choix	Deuxième choix	Troisième choix
b	50*5= 250		
c	25*5= 125	25*2= 50	25*1= 25
d	80*2= 160	30*2= 60	30*2= 60
e	70*3= 210	20*3= 60	20+20+10= 50
f	60*2= 120	60+10= 70	
g	99*1= 99	49*1= 49	49*1= 49
h	90*1= 90	40*1= 40	30*1= 30



Le bénéfice des vues à chaque itération.

Exemple (2)

- **Premier choix** : initialement l'ensemble S ne contient que la vue sommet a donc touts les calculs du bénéfice se font à partir de cette vue. Si on prend la vue b pour être matérialisée, on réduit son coût et celui de ses descendants (d , e, g ,h) de 50, donc le bénéfice de B (b, S) = (100-50)*5 = 250. Le même traitement se fait avec toutes les autres vues. Si on prend la vue e pour être matérialisée, on réduit son coût ainsi que le coût de ses descendants (g et h) de 70 donc le bénéfice de e est B(e, S) = (100-30)*3 = 70*3 = 210.

  -Pour le premier choix, on trouve que b a le plus grand bénéfice B(b, S) = 250, b est donc considérée comme la première vue matérialisée après a (la vue sommet).

  S = {a, b}.

- **Deuxième choix** : Trouver le deuxième choix consiste à refaire la même opération précédente pour toutes les vues, avec S = {a, b}.
- Par exemple, choisir f réduit son coût de 60 (de 100 qui est le coût de a à 40 qui est le coût de f), la vue f a une seule vue descendante h. h dépend de la vue matérialisée b (best la vue qui a le plus petit coût dans S). Donc Bh = c(u) – c(v) = c(b) – c(f) = 50 – 40 = 10.
- Le bénéfice de f est alors B (f, S) = Bf + Bh = 60 + 10 = 70. Comme il est indiqué dans le tableau 4.1 f a le plus grand bénéfice, elle est donc considérée comme la deuxième vue à matérialiser. S = {a, b, f}.

Exemple (3)

- **Troisième choix :** le troisième choix est indiqué dans la troisième colonne du tableau Les mêmes calculs que la première et la deuxième étape se font, mais en considérant l'ensemble S = {a, b, f}. Comme exemple d'un calcul un peut compliqué le calcul du bénéfice de e.
- Le choix de e réduit son coût de 20. Be = c(u) – c(v) = c(b) – c(e) = 50 – 30 = 20 (On a pris b car c'est l'ancêtre matérialisé de e qui a le plus petit coût dans S).
- La vue e a deux vues descendantes g et h, leurs bénéfices sont calculés comme suit :

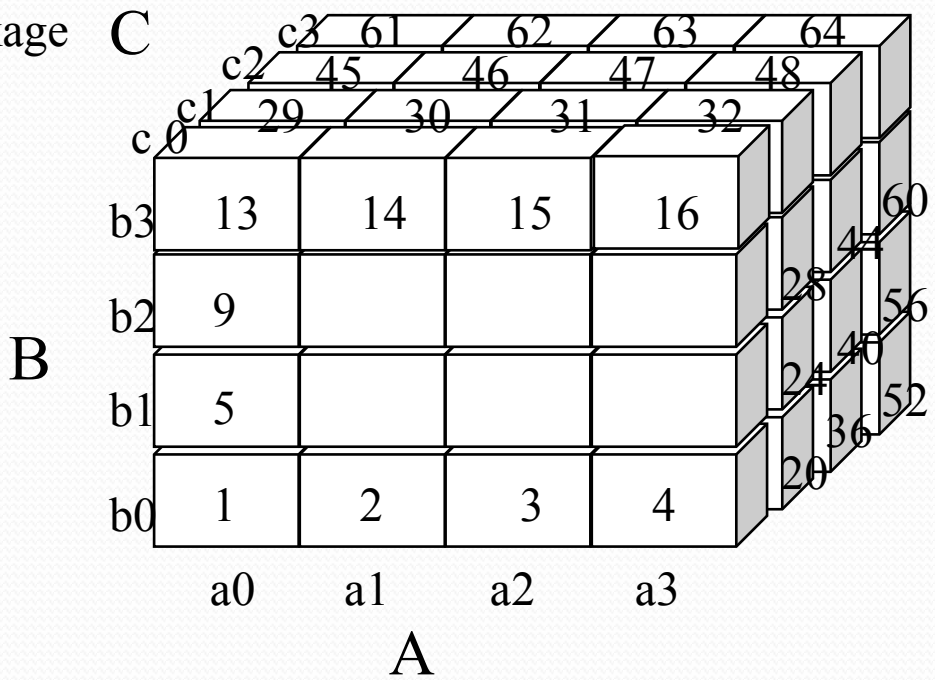
  B_h = c(u) – c(v) = c(f) – c(e) = 40 – 30 = 10 (On a pris f car c'est l'ancêtre matérialisé de h qui a le plus petit coût dans S).

  Bg = c(u)- c(v) = c(b) – c(e) = 50 – 30, (b est l'ancêtre matérialisé de g qui a le plus petit coût dans S) Bg = 20. Donc le bénéfice de e est :

  $B(e, S) = \Sigma_{W \leq e} Bw = Be + B_h + B_g = 20 + 10 + 20 = 50.$
- Après avoir calculé tous les bénéfices des vues restantes, d est sélectionné pour la matérialisation avec un bénéfice égal à 60.
- L'ensemble des vues sélectionnées est donc S = {a, b, f, d}.
- Coût total = 100 * 8 = 800. (100 est le coût de la vue a, 8 est le nombre de vues)
- Bénéfice total = B (b, S) + B (f, S) + B (d, S) = 380.
- Cet ensemble réduit le coût de 800 (le cas où seulement la vue a est matérialisée) à 420. (800 – 380 = 420).

Matérialisation totale: Multi-way Array Aggregation for Cube Computation

- Partitionner les tableaux en des portions (chunk : un petit sous-cube qui peut être chargé en mémoire)
- Adressage de tableaux creux compressés : (chunk_id, offset)
- Calculer les agrégats en “multiway” en visitant les cellules du cube de sorte à (i) minimiser le # de fois que chaque cellule est visitée et (ii) réduire les coûts d’accès et de stockage



Quel est l’ordre préférentiel pour parcourir le cube dans le cadre d’une agrégation?

Exemple (1)

- On doit calculer les cuboïdes A, B, C, AB, BC, AC, ABC et Ø
- Supposons que les dimensions A,B et C ont les tailles 40, 400, 4000. La taille de chaque partition de A,B et C est donc 10, 100 et 1000. Ex: A:magasins, B: Date,C: Produit et la mesure: #produits vendus
- La portion $a_0b_0c_0$ correspond à 1, $a_1b_0c_0$ correspond à 2, …
- Supposons que l'on veuille calculer le cuboïde BC. Ceci peut se faire en calculant les cuboïdes b_ic_j .
- Pour calculer b_0c_0, il faut parcourir les portions 1,2,3,4. Pour b_1c_0, on lit 5,6,7,8. Au total, Pour BC, on doit scanner les 64 portions.
- Pour calculer les cuboïdes AC et AB, faut-il rescanner les 64? NON
  - Quand la portion 1 ($a_0b_0c_0$) est scannée, on peut en profiter pour entamer le calcul de a_0b_0 et a_0c_0 d'où le terme Multi-way aggregation.
  - Donc, en scannant les 64 portions une seule fois chacune, on peut calculer les 3 cuboïdes AB, AC et BC

EXEMPLE (2)

- Les tailles de AB, AC et BC sont resp. 16.10^3, 16.10^4 et 16.10^5

- En parcourant les portions de 1 à 64 dans cet ordre, b_0c_0 est calculé en lisant 1,2,3,4 ; b_1c_0 calculé en lisant 5,6,7,8 …

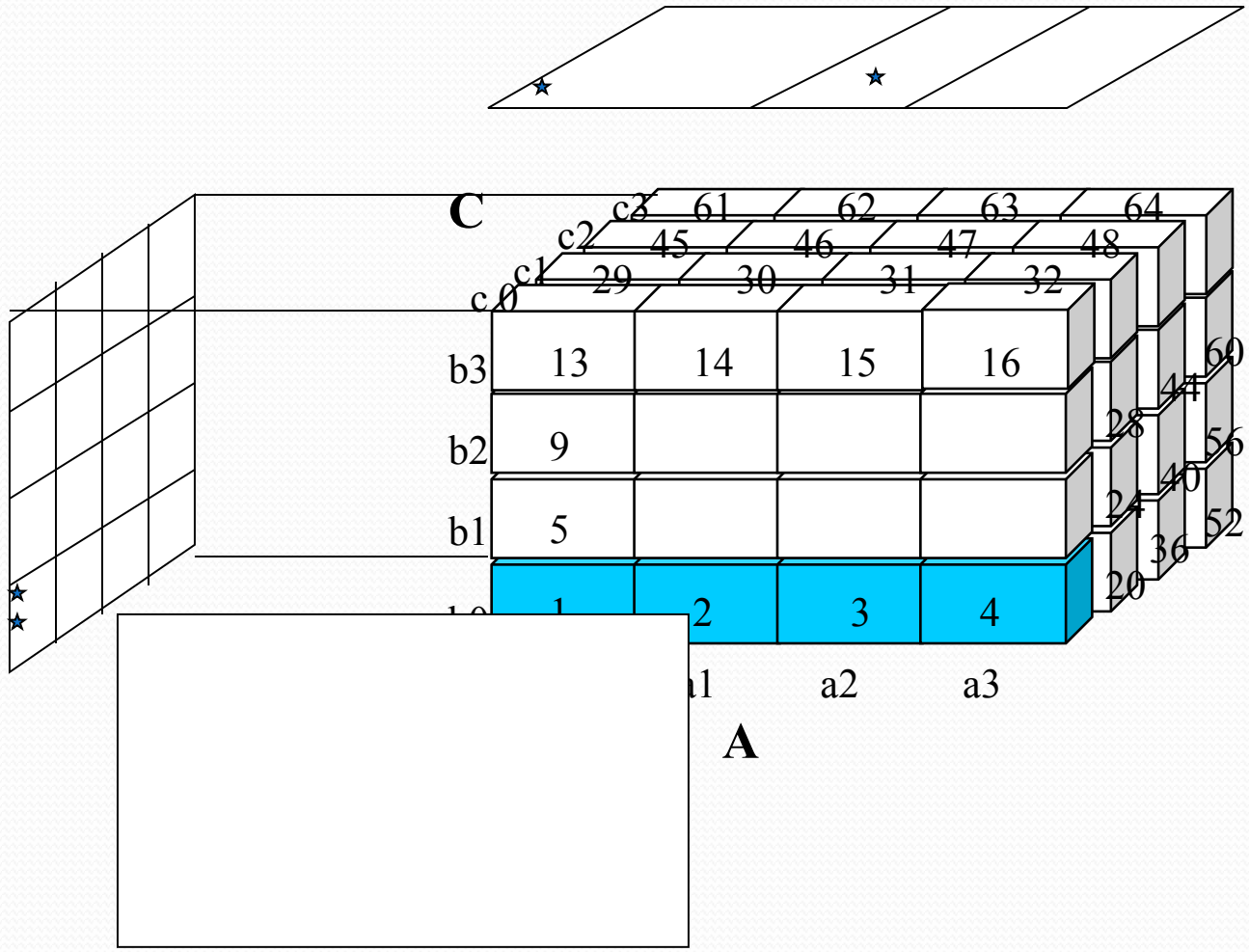
  a_0b_0 est calculé en utilisant 1, 17, 33 et 49 (49 portions)

  a_0c_0 est calculé en utilisant 1, 5, 9 et 13 (13 portions)

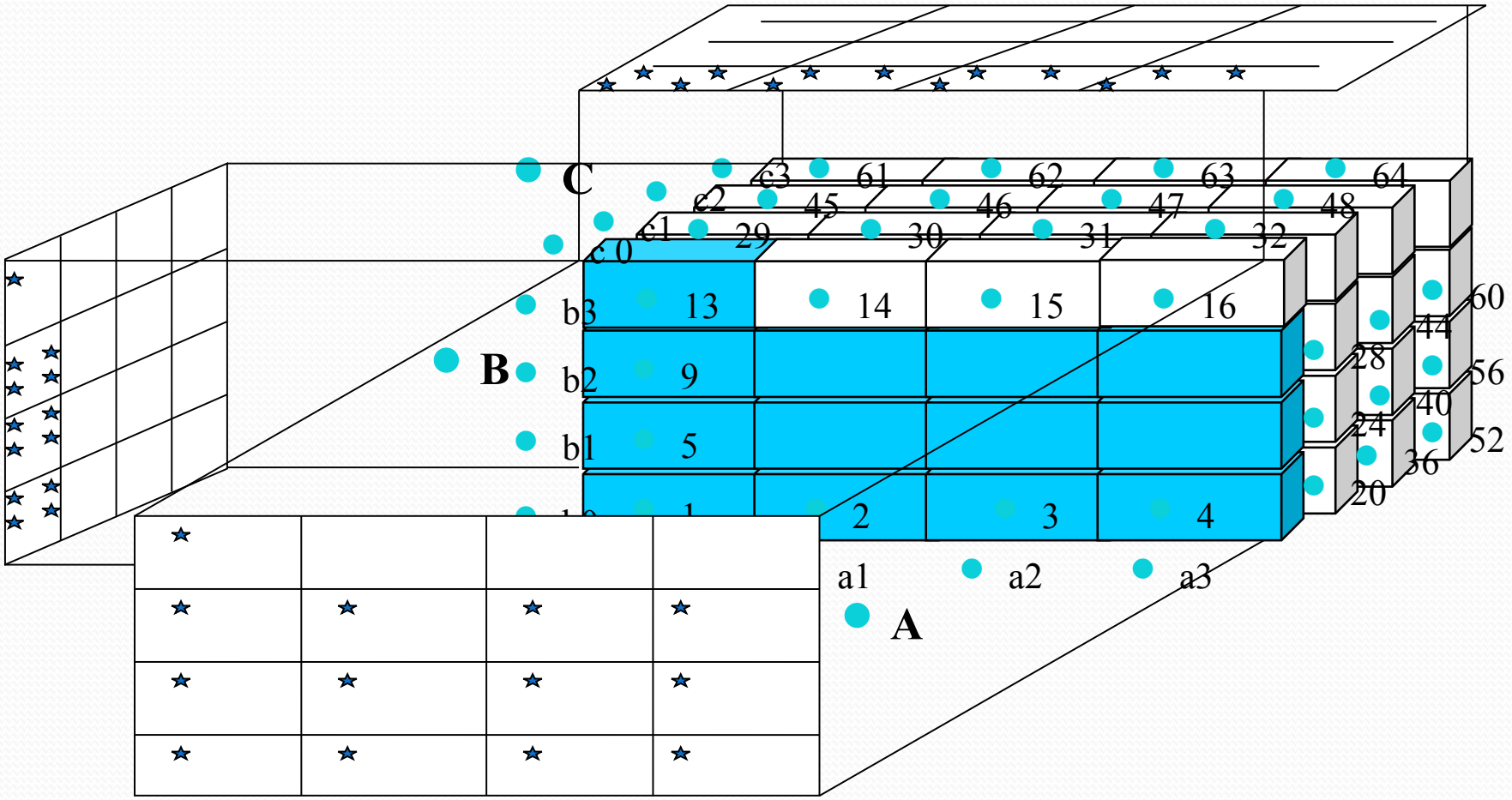
- Pour éviter qu'une portion ne soit chargée plus d'une fois, l'espace tampon minimum doit être: 40*400 (plan AB)+10*4000 (une ligne du plan AC) + 100*1000(une portion de BC)=156 000
- Supposons que l'ordre de parcours est 1, 17, 33, 49, 5, 21, 37, 53, … agrégation selon AB, puis AC puis BC. L'espace mémoire requis est 400*4000(plan BC)+40*4000(une ligne de AC)+10*100(portion de AB)=1 641 000

- Conclusion: L'ordre de prise en compte des dimensions est important. Il faut trier les plans dans l'ordre croissant de leur taille (AB est le plus petit) puis parcourir les portions en tenant compte de cet ordre.

Multi-way Array Aggregation for Cube Computation



Multi-way Array Aggregation for Cube Computation



Pas de matérialisation: Indexation de données OLAP: Index Bitmap

- Index sur une colonne particulière
- Chaque valeur dans la colonne a un vecteur de bits : opérations sur les bits sont rapides
- La longueur du vecteur de bits: # de la table de base
- Le i-ème bit=1 si la i-ème ligne de la table de base *possède la valeur de la colonne indexée*
- Ne convient que pour les domaines de faible cardinalité

Table de base

Cust	Region	Type
C1	Asie	Gross
C2	Europe	Détaill
C3	Asie	Détaill
C4	Amériqu	Gross
C5	Europe	Détaill

Index sur Région

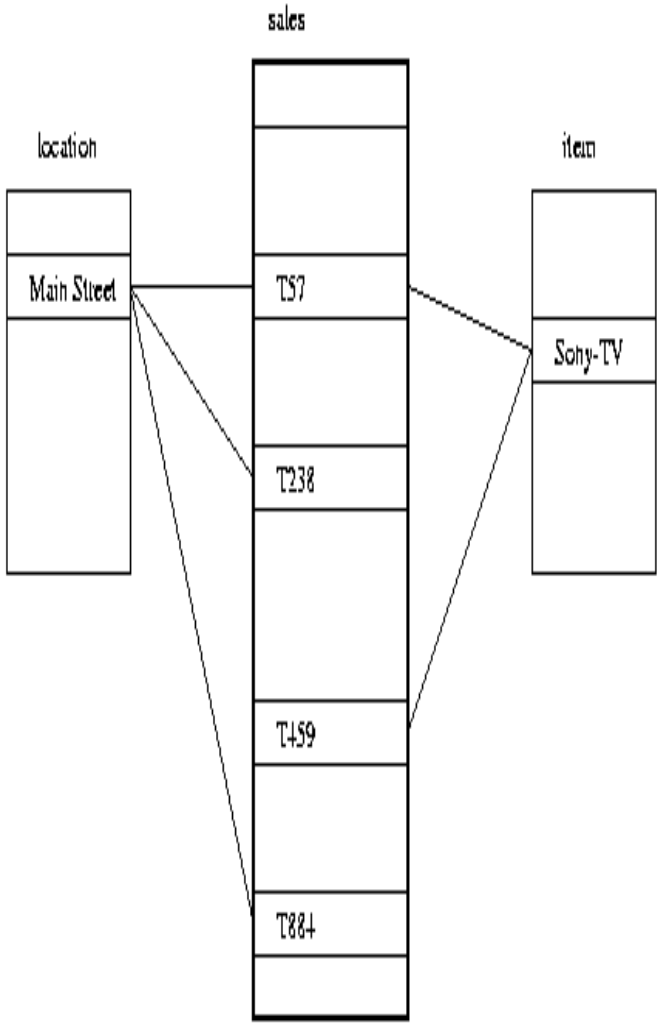
RowId	Asie	Europe	Amérique
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1
5	0	1	0

Index sur Type

RecID	Gross	Détaill
1	1	0
2	0	1
3	0	1
4	1	0
5	0	1

Indexing OLAP Data: Index de jointure

- Index de jointure : JI(R-id, S-id) où
  R (R-id, …) ⊳⊲ S (S-id, …)
- En général, un index associe une valeur à une liste d'Id d'enregistrements
  - matérialise la jointure dans un fichier JI pour accélérer l'exécution de la jointure
- Dans les data warehouses, un index de jointure relie des valeurs de dimensions à des lignes de la table de faits.
  - Ex. Table de faits: *Sales* et 2 dimensions *ville* et *produit*
    - Un index de jointure sur *ville* maintient pour chaque ville une liste de ID-enreg's de tuples concernant des ventes dans la ville en question
  - Les index de jointure peuvent être partagés par plusieurs dimensions



Traitement efficace des requêtes OLAP

- Déterminer quelles sont les opérations à effectuer sur les cuboïdes disponibles :
  - transformer drill, roll, etc. en des requêtes SQL et/ou opérations OLAP, ex, dice = sélection + projection
- Déterminer sur quel cuboïde matérialisé l'opération doit être exécutée.
- Exploiter les structures d'index disponibles

Résumé

- Data warehouse
  - Données orientées sujet, intégrées, dépendant du temps, et non-volatiles pour la prise de décision
- Un modèle multidimensionnel pour les data warehouses
  - Schéma en étoile, schéma snowflake, constellation
  - Un data cube est decrit par des dimensions et des mesures
- Opérations OLAP : drilling, rolling, slicing, dicing and pivoting
- OLAP servers: ROLAP, MOLAP, HOLAP
- Implémentation des data cubes
  - Matérialisation Partielle vs. totale vs. nulle
    - Sélection des cuboïdes à matérialiser
    - Multiway array aggregation
    - Implémentation avec Bitmap index et join index

FIN