Université Pierre et Marie Curie - Master d'Informatique

Saison février - juin 2014

MI034 : Bases de Données Réparties
TD et TME

Stéphane GANCARSKI, Hubert NAACKE

URL : http://www-bd.lip6.fr/ens/bdr2014/index.php/Accueil

TABLE DES MATIERES

TD1 Techniques d'indexation

Rappel : Arbres B+

Un arbre B+ est composé de noeuds tels que :
Le noeud racine contient au moins une clé. Un noeud intermédiaire contient c clés avec $d \leq c \leq 2 * d$ (d est fixé en fonction l'espace alloué pour un noeud). Seules les feuilles font référence aux données indexées. Le nombre de noeuds à traverser pour aller de la racine vers une feuille est constant (égal à la profondeur de l'arbre). .

1. Soit un noeud contenant c clés, combien contient-il de références vers ses noeuds fils ?
2. Soit un noeud contenant n clés $\{c_1, ..., c_n\}$ et $n+1$ références $\{r_1, ..., r_{n+1}\}$. On représente un noeud par la liste $(r_1, c_1, ..., r_i, c_i, ..., r_n, c_n, r_{n+1})$. Soit K_i l'ensemble des clés contenues dans le sous arbre référencé par r_i. Les clés de K_i sont comprises dans l'intervalle $[a, b[$. Quelles sont les valeurs a et b pour K_1, K_i et K_{n+1} ?

Exercice 1 : Arbres B+

On considère un arbre B+ dont les nœuds et les feuilles peuvent contenir au plus 4 clés et au minimum (sauf la racine) 2 clés. A l'état initial, la racine de l'arbre contient la clé 50. Un seul niveau intermédiaire contient (tous nœuds confondus) les clés 8, 18, 32, 40 73, 85. Les feuilles contiennent (toutes feuilles confondues) les valeurs 1, 2, 5, 6, 8, 10, 18, 27, 32, 39, 41, 45, 52, 58, 73, 80, 91, 99.

1. Dessinez l'état inital de l'arbre.
2. Montrer l'état de l'arbre après l'insertion de la clé 9.
3. Montrer l'état de l'arbre résultant de l'insertion dans l'arbre original de la clé 3. Combien de lecture/écriture de page cette insertion va-t-elle nécessiter ?
4. On considère la redistribution éventuelle des clés avec les feuilles voisines **de même père**. En considérant uniquement la redistribution à gauche, montrer l'état de l'arbre résultant de la suppression dans l'arbre original de la clé 8. Reprendre cette question en considérant uniquement la redistribution à droite.
5. Montrer l'état de l'arbre résultant, à partir de l'arbre original, de l'insertion de la clé 46 suivie de la suppression de la clé 52 (fusion éventuelle avec le voisin de droite).
6. Montrer l'état de l'arbre résultant, à partir de l'arbre original, de la suppression successive des clés 32, 39, 41, 45 et 73.

Exercice 2 : Arbres B+

On considère un arbre B+ dont les nœuds et le feuilles peuvent contenir au plus 4 clés et au minimum (sauf la racine) 2 clés. A l'état initial, la racine de l'arbre contient les clés 13, 17, 24, 30. Les feuilles contiennent (toutes feuilles confondues) les valeurs2, 3, 5, 7, 14, 16, 19, 20, 22, 24, 27, 29, 33, 34, 38, 39. On considère uniquement la redistribution à gauche.

1. Dessiner l'état initial de l'arbre.
2. Donner 4 valeurs de clé telles que leur insertion successive puis leur suppression dans l'ordre inverse résulte dans un état identique à l'état initial.
3. Donner une valeur de clé dont l'insertion suivie de la suppression résulte dans un état différent de l'état initial.
4. Combien au minimum faut-il insérer de clé pour que l'arbre gagne deux niveaux en hauteur ?

Exercice 3 (facultatif) : Arbres B+

On suppose qu'on dispose d'un fichier trié et qu'on veut construire un arbre B+ pour ce fichier. La méthode la plus simple pour cela est d'utiliser l'algorithme d'insertion et d'insérer ls clés les unes après les autres. Quels sont les inconvénients de cette approche ? Proposer une autre approche qui remédie à ces problème.

Exercice 4 : hachage statique

Rappeler les principes du hachage statique. Quels en sont les inconvénients lorsqu'un fichier diminue ou augmente beaucoup. Quelle est la seule solution pour y remédier ? Peut-elle être utilisée à tout moment ?

Exercice 5 : hachage dynamique (extensible)

Une structure de hachage extensible contient des entrées $x*$ dont les valeurs de hachage $h(x)$ sont les suivantes : 1, 4, 5, 7, 10, 12, 15, 16, 19, 21, 32.

1. Représenter ces valeurs sous forme binaire.
2. Avec au plus 4 entrées par paquets, combien faut-il de paquets pour avoir la structure la plus compacte possible ?
3. Dessiner la structure de hachage pour ces valeurs (répertoire et paquets, en indiquant profondeur globale et profondeurs locales)
4. Expliquer pas à pas comment retrouver le paquet contenant l'entrée $7*$ (tel que $h(x) = 7$).
5. Insérer $13*$
6. Insérer $20*$
7. Insérer $29*$
8. Donner un exemple de plus petit ensemble de clés à supprimer pour obtenir une division par 2 du répertoire.

Exercice 6 : hachage extensible

On considère une structure de hachage extensible de profndeur globale 3. Dans l'état initial, les entrées du répertoire sont les suivantes : (000, *paquetA), (001, *paquetB), (010, *paquetC), (011, *paquetD), (100, *paquetA2), (101, *paquetB), (110, *paquetC), (111, *paquetD). (000, *paquetA). Les entrées de données sont les suivantes : $1*, 4*, 5*, 7*, 10*, 12*, 15*, 16*, 20*, 21*, 36*, 51*, 64*$.

1. Représenter la structure complète dans son état initial.
2. Que peut-on dire de la dernière valeur insérée dans cette structure ?
3. Même question si on suppose qu'il n'y a pas eu de suppression depuis la dernière insertion. Cette insertion a-t-elle causé un éclatement ? En cas de réponse négative, que peut-on dire de la dernière insertion ayant causé un éclatement (on suppose toujours qu'il n'y a pas eu de suppression depuis).
4. Inśerer $68*$ dans la structure initiale.
5. Inśerer $17*$ et $69*$ dans la structure initiale.
6. Supprimer $21*$ de la structure initiale.
7. Supprimer $10*$, puis $64*$ et enfin $16*$ de la structure initiale.

Exercice 7 (facultatif) : hachage extensible

Montrer un exemple de structure de hachage extensible où l'insertion de trois entrées provoque trois éclatements et leur suppression (dans l'ordre inverse) remet la structure dans son état initial.

Exercice 8 : Comparaison des méthodes d'accès

On considère une relation $R(a, b, c, d)$ contenant 1000000 de nuplets. Chaque page de donnée peut contenir 10 nuplets. a est une clé candidate de R ($a \in [0, 1000000[$) et les tuples de R sont stockés sur le disque sans être triés. Il y a M clés par feuille. On a le choix entre 3 méthodes d'accès :

- Parcourir séquentiellement le fichier (non trié) contenant R.
- Utiliser un arbre-B+ sur $R.a$ (on suppose la hauteur de cet arbre égale à 3)
- Utiliser une table de hachage sur $R.a$ (on suppose que les paquets ont un taux d'occupation de c)

On suppose que dans les 2 cas (arbre B+ et hachage), lorsqu'on accède une entrée de donnée dans la structure, il suffit d'une lecture de page pour accéder ensuite au nuplet désiré.

Pour chacune des requêtes suivantes, évaluer le nombre d'E/S nécessaires pour chaque méthode et déduire quelle est la meilleure méthode.

1. trouver tous les nuplets de R.
2. trouver tous les nuplets de R tels que $a < 50$
3. trouver tous les nuplets de R tels que $a = 50$
4. trouver tous les nuplets de R tels que $a > 50$ et $a < 100$

Extrait du partiel BDA (avril 2002)

Exercice 2 : Profondeur des arbres B+ pts

Soit la relation R(a1, a2, a3) contenant $\mathbf{10^7}$ tuples,
L'attribut a1 est la clé primaire, il est indexé par un arbre B+.
L'index est dense : toute valeur distincte de a1 correspond à une clé d'une feuille de l'arbre.

Question 1

Le nombre de clé par nœud est *nc* tel que :
$nc \in [4,9]$ pour tout nœud sauf la racine, $nc \in [1,9]$ pour la racine
La profondeur *p* d'un arbre correspond au nombre de niveaux, racine incluse, soit:
$p = 1$ pour un arbre réduit à sa seule racine,
$p = 2$ pour un arbre ayant seulement une racine et des feuilles,
etc…
Quelle est la profondeur minimale de l'index sur a1? Expliquez brièvement votre réponse.

Question 2

Soit la requête R1 : *Quelle est la valeur de a2 pour un a1 donné de valeur x ?*
R1(x) : select a2 from R where a1 = x
Pour améliorer les performances de la requête R1, la structure de l'index sur a1 est modifiée :
Le nombre maximal de clés par noeud est modifié,
La valeur de a2 est stockée directement dans les feuilles de l'arbre : chaque feuille contient un ensemble de couples (clé, a2). Il n'y a pas de liste chaînée entre les feuilles.
L'attribut a2 est une chaîne de 50 octets. La taille d'une clé est 4 octets. La taille d'une référence à un noeud est 8 octets. La taille d'un nœud est de 4000 octets.

Quel est le nombre maximum de couples (clé, a2) dans une feuille ?
Quel est le nombre maximum de clés dans un nœud intermédiaire ?
Quelle est la profondeur minimale de l'index sur a1 ?

Exercice 3 : Arbres B+ : insertion et suppression de clés pts

Soit un arbre B+ dont le nombre de clé par nœud est *nc* tel que :
$nc \in [2,4]$ pour tout nœud sauf la racine, et $nc \in [1,4]$ pour la racine.
a) Dessiner l'arbre **A1** ayant 2 niveaux tels que :
- la racine a les clés 10, 20, 23, 40.
- les feuilles ont les clés 1, 2, 4, 6, 10, 12, 14, 20, 22, 24, 30, 32, *c*, 42 avec $(32 < c < 42)$.

b) Donner toutes les valeurs possibles pour la clé *c*.
c) Représenter l'arbre **A2** après insertion de la clé 8 dans A1, sans redistribuer les clés avec les voisins. En cas d'ajout d'un nœud, le nœud créé doit contenir le nombre minimal de clés.

d) Représenter l'arbre **A3** après suppression de la clé 8 dans A2. On considère la redistribution éventuelle avec le voisin de gauche d'abord.
e) Combien de clés au maximum peut on supprimer dans l'arbre **A3** sans qu'il perde un niveau ? Donner un exemple de clés que l'on peut supprimer.

Exercice 4 : Fonction de hachage multi-attributs pt

Soit la relation Livre(num, mois, année, titre). Les domaines sont : année $\in$ [1900-1999], mois $\in$ [1-12]. La distribution des attributs est uniforme. Les livres sont indexés avec une seule table de hachage sur les deux attributs mois et année. Proposer une fonction de hachage h1(mois, année) pour répartir uniformément les tuples dans 1200 paquets.
Proposer une fonction h2(mois, année) pour répartir uniformément les tuples dans 512 paquets.

Extrait du partiel BDR (avril 2003)

Exercice 1 : Arbre B+ 4 pts

Soit un arbre B+ avec 3 niveaux. Le nombre de clés par nœud est tel que :
la racine et les nœuds intermédiaires contiennent de **1** à **2** clés,
les feuilles contiennent de **2** à **3** clés,

Question 1 :

1.1) Dessiner l'arbre **A1** ayant 3 niveaux tels que :
Les feuilles ont les clés 1, 4, 9,16, 25, 36, 49, 54, 61, 70, 81, 84, 87, 88, 95, 99.
Le niveau intermédiaire a les clés 9, 54, 70, 88
La racine contient 2 clés (pour les clés de la racine: choisir les **plus petites** valeurs possibles **parmi les clés des feuilles**).

1.2) Représenter l'arbre A2 après insertion de la clé 32 dans A1, sans jamais redistribuer de clé avec les voisins.

Créer un nouveau nœud en cas de débordement d'une feuille ou d'un nœud intermédiaire.

1.3) Soit l'arbre A3 après insertion de la clé 32 dans **A1**. L'arbre A3 est obtenu en redistribuant si possible les clés des niveaux intermédiaires. Quel est le nombre de nœuds de l'arbre A3 (racine, nœuds intermédiaires et feuilles inclus) ?

1.4) Représenter l'arbre A4 après suppression de la clé 16 dans **A1**. L'arbre A4 est obtenu en redistribuant si possible les clés des feuilles avec les voisins.

1.5) Quel est le nombre minimum de clé à supprimer dans A1 pour qu'il perde un niveau ? Justifier votre réponse.

Donner un exemple de clés à supprimer (en choisissant des valeurs de clé les plus petites possibles).

Question 2 :
Soit la relation **Produit**(<u>numéro</u>, prix). Les produits sont stockés dans l'ordre croissant du numéro. L'attribut *numéro* est une clé, il est indexé par un arbre B+. Les attributs sont indépendants et leur distribution est uniforme.
La relation *Produit* a 100 000 n-uplets, une page de données contient 100 n-uplets.
L'index est non dense (i.e., une seule clé par page de données).

2.1) Quel est le nombre minimum de clés au niveau des feuilles ? Justifier votre réponse.

2.2) Quel est le nombre minimal de niveaux de l'index ? Justifier votre réponse.

Extrait du partiel BDR (avril 2004)

Exercice 1 : Indexation par hachage (2004) 5 pts

On considère un index utilisant une technique de hachage extensible. La profondeur globale est indiquée au dessus du répertoire. La profondeur locale est indiquée au dessus de chaque paquet. Un paquet peut contenir de 1 à 4 entrées. Les entrées sont des nombres entiers strictement positifs.

On utilise un algorithme de suppression complète, c'est-à-dire que la suppression d'un paquet entraîne la diminution de la taille du répertoire (division par 2), si cela est possible. On suppose qu'il y a fusion seulement si un paquet est vide.

Question 1.

Soit T la taille du répertoire de l'index. L'index contient au maximum T paquets, et 4*T entrées

a) Quel est le nombre minimum de paquets de l'index ? Justifier et donner la profondeur locale des paquets.

b) Soit Nmin est le nombre minimum d'entrées de l'index. Que vaut Nmin ?

c) Soit un index avec Nmin entrées dont les valeurs sont les plus petites possibles. Quelle est la valeur de la plus grande entrée de l'index ?

Question 2. On considère l'état suivant d'un index utilisant une technique de hachage extensible.

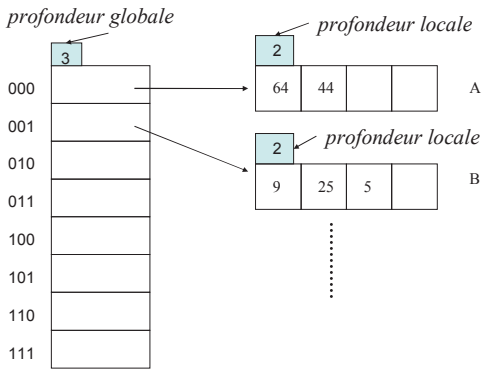
a) Complétez l'état initial de cet index de telle façon que :

L'état initial est obtenu par plusieurs insertions, sans aucune suppression. Vous devez déterminer des valeurs d'entrées à insérer, les plus petites possibles.

L'index dans l'état initial doit contenir une entrée telle que sa suppression ultérieure provoquera une diminution de la profondeur globale.

b) Donnez l'état de l'index après suppression de cette entrée.

Index à compléter :



Clé à supprimer : ...

Index après suppression :

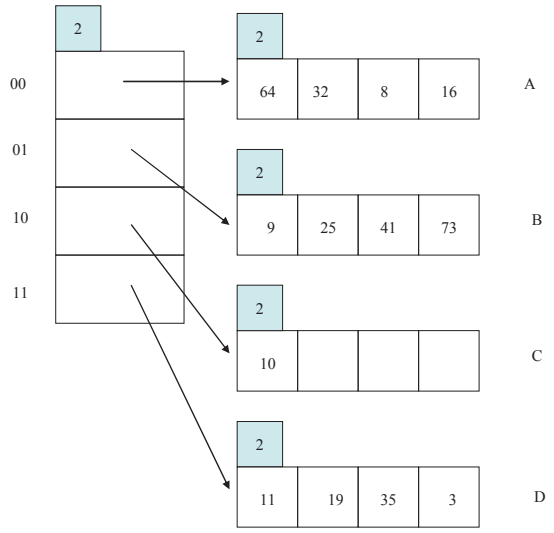
Question 3. Soit l'index suivant dans l'état initial *E1*.

Donnez une liste de trois valeurs e1, e2, e3, d'entrées d'index, telle que :

- leur insertion (dans l'ordre e1, e2, e3) dans l'index ci-dessous provoque 3 éclatements,
- puis leur suppression dans l'ordre inverse (e3, e2, e1) redonne l'index initial *E1*.

Représenter l'état *E2* de l'index après insertion de ces trois valeurs.

Index initial (E1):



Entrées à insérer : e1=….. e2=….. e3=…..

Index après insertion (E2):

Question 4. On veut modifier les valeurs initiales de l'index E1 de telle sorte que :

- l'insertion dans l'index (dans l'ordre e1, e2, e3) des entrées déterminées à la question précédente, provoque 3 éclatements,
- la suppression de ces trois entrées dans l'ordre inverse de leur insertion (supprimer d'abord e3, puis e2, puis e1) ne provoque **pas** de suppression de paquet.

Quel est le nombre minimum de valeurs d'entrées à modifier ?

Représenter l'index E1' dans l'état initial modifié. Souligner les valeurs modifiées. Si vous avez le choix parmi plusieurs entrées à modifier, modifier l'entrée la plus petite.

Exercice 2 : Arbre B+ (partiel 2004) 4 pts

On considère des arbres B+ contenant des données de type entier, tel que les nœuds et les feuilles contiennent au plus quatre valeurs. Les noeuds (sauf la racine) et les feuilles doivent être au moins à moitié pleins (2 valeurs au moins). La hauteur d'un arbre est égale à son nombre de niveaux. Un arbre de hauteur 1 est réduit à sa seule racine.

Question 1. Donnez un exemple d'arbre B+ dont la hauteur passe de 2 à 3 lorsqu'on y insère la valeur 25. Donnez les deux arbres, avant et après l'insertion. Utiliser la trame quadrillée pour dessiner les nœuds et laisser un espace entre deux nœuds. Répondre page suivante.

Question 2. Donnez un exemple d'arbre B+ dans lequel la suppression de la valeur 25 conduit à une redistribution. Donnez les deux arbres, avant et après la suppression.

Question 3. Donnez un exemple d'arbre B+ dans lequel la suppression de la valeur 25 conduit à une fusion de deux nœuds, mais ne modifie pas la hauteur de l'arbre. Donnez les deux arbres, avant et après la suppression.

Exercice 3 : Arbres B+ (partiel 2005) 4 pts

On considère un arbre B+ tel qu'un nœud quelconque peut contenir de **1** à **3** clés.

Question 1

a) Dessiner l'arbre résultant de l'insertion successive des 10 clés 1, 2, ..., 10, dans l'ordre croissant, dans un arbre initialement vide. En cas de débordement d'une feuille, l'éclater en 2 feuilles ayant chacune le même nombre de clés. Utiliser la trame quadrillée pour dessiner les nœuds et laisser un seul espace entre deux nœuds.

b) On considère l'insertion successive de N clés 1, 2, ..., N, dans l'ordre croissant, dans un arbre initialement vide. Donner la valeur maximale de N lorsque l'arbre obtenu est de profondeur 3 (un seul niveau intermédiaire).

c) On considère l'arbre de profondeur 3 qui a le nombre maximum de clés (sans se préoccuper de la façon d'obtenir cet arbre), et M le nombre total de clés dans ses feuilles. N_{max} est il égal à M ? Justifier.

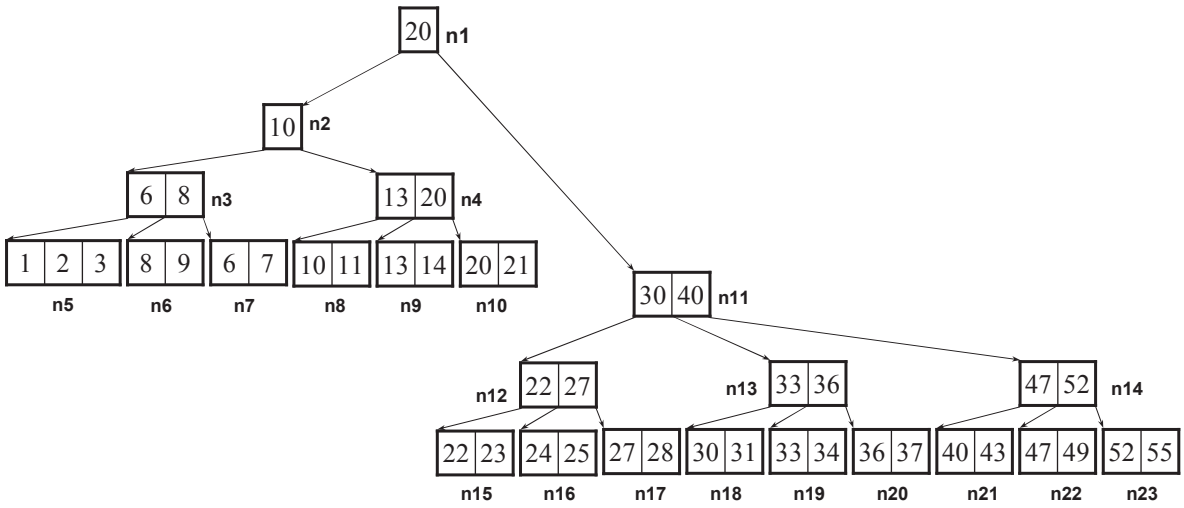
Question 2

Donner une suite de 10 clés à insérer successivement, dans un arbre vide afin d'obtenir un arbre de profondeur 2 (pas de niveau intermédiaire), en choisissant la valeur des clés parmi les entiers dans [1, 10]. Puis représenter l'arbre obtenu.

Indication : Répondre de manière à insérer autant que possible les plus petites valeurs en premier. Parmi toutes les réponses possibles, donner celle qui est la plus "proche" d'une suite croissante. C'est-à-dire, donner la suite qui est composée d'un nombre minimum de sous-suites croissantes.

Exercice 4 : Arbres B+ (partiel 2005) 2 pts

On considère l'arbre B+ d'ordre 2 suivant :



Trouver toutes les erreurs dans cet arbre. Indiquer le numéro n_i du noeud erroné et expliquer brièvement l'erreur. S'il est possible de corriger l'erreur sans restructurer l'arbre, mais en modifiant seulement des valeurs de clés, alors suggérer une correction.

TD 2a - Optimisation de requêtes

Généralités

1.Décrire l'avantage d'un plan d'exécution en pipeline pour un arbre de jointures

2.Pendant l'optimisation de requête, quel est le rôle des statistiques collectées sur les données de la base ?

Exercice 1 : Attributs à lire pour traiter une requête

1. Soit le schéma relationnel:

Etudiant(num, integer, dnum: integer, nom:char(20), âge: integer)

Dept(dnum: integer, matière: char(20), salle: integer, budget: real)

Pour chaque requête SQL suivante, donner la liste des attributs qui doivent être lu pour obtenir le résultat de la requête.

1.SELECT * FROM Etudiant
2.SELECT * FROM Etudiant, Dept
3.SELECT * FROM Etudiant E, Dept D WHERE E.dnum = D.dnum
4.SELECT E.num, D.matière FROM Etudiant E, Dept D WHERE E.dnum = D.dnum
5.SELECT COUNT(*) FROM Etudiant E, Dept D WHERE E.dnum = D.dnum
6.SELECT MAX(E.age) FROM Etudiant E, Dept D WHERE E.dnum = D.dnum
7.SELECT MAX(E.age) FROM Etudiant E, Dept D WHERE E.dnum = D.dnum AND D.salle = 5
8.SELECT E.dnum, COUNT(*) FROM Etudiant E, Dept D WHERE E.dnum = D.dnum GROUP BY D.dnum
9.SELECT D.salle, AVG(D.budget) FROM Dept D GROUP BY D.salle HAVING COUNT(*) > 2
10.SELECT D.salle, AVG(D.budget) FROM Dept D GROUP BY D.salle ORDER BY D.salle

Exercice 2 : Modèle de coût

L'objectif de cet exercice est de comparer deux méthodes pour le choix du plan d'exécution d'une requête. La première méthode est la transformation de plan basée sur des heuristiques. La 2ème méthode est la génération exhaustive de l'espace de recherche associée au choix du plan candidat de moindre coût.

Soit le schéma R1(A,B, …), R2(A,B, …) et R3(A, B, …)

Soit la requête :
```
select R1.A, R3.B   from R1, R2, R3
where R1.A=R2.A and R2.A=R3.A and R1.B=1 and R2.B=2
```

1) Donner l'arbre algébrique correspondant en respectant l'ordre des prédicats donnés dans la clause where.
2) Donner un arbre équivalent en appliquant les opérations les plus réductrices (restriction, projection) d'abord.
3) Soit le modèle de coût simplifié suivant, où l'unité de coût est l'accès à un tuple.

- Pour toute relation R, si card(R) est le nombre de tuples de R, le coût d'une *restriction* sur égalité est :

  coût ($\sigma_{att.=valeur}(R)$) = card($\sigma_{att.=valeur}(R)$) s'il y a un index sur att. (arbre-B+ ou hachage),
  = card(R) sinon.

- Le coût d'une lecture séquentielle est le coût d'une restriction sans index.
- Pour toutes relations R ayant card(R) tuples et S ayant card(S) tuples, le coût d'une *equi-jointure* est :

  coût ($R |><|_{att.} S$) = card(R) s'il y a un index sur l'attribut att. de S
  = card(R)*card(S) sinon.

- Le coût d'un opérateur traité en pipeline est nul.

- On suppose que R1, R2, R3 ont les caractéristiques suivantes :
  - il y a un **index** sur l'attribut B de R1, A est clé primaire de R1, R2 et R3 (il existe un index plaçant pour chaque clé primaire)
  - card(R1) = card(R2) = card(R3) = 1000, $\pi_A(R1) = \pi_A(R2) = \pi_A(R3)$
  - il y a 10 valeurs possibles pour B, uniformément réparties dans R1 et aussi dans R2 et dans R3.
  - pour chaque relation, la répartition des valeurs de A en fonction de B est uniforme et est indépendante de la répartition dans les deux autres relations

Questions :

a) Quelle est la cardinalité du résultat de la requête ?

b) Donner l'arbre algébrique de coût minimal et son coût (le nombre total d'accès à des tuples), en ignorant les projections. Préciser votre réponse en détaillant le coût et la cardinalité des résultats intermédiaires.

Exercice 3 : Espace de recherche

Soit le schéma relationnel :

Joueur(licence: integer, cnum : integer, salaire: integer, sport: char(20))

Club(cnum: integer, nom: char(20), division: integer, ville : char(10))

Finance(cnum: integer, budget: real, dépense: real, recette: real)

On considère la requête:

```
SELECT C.nom, F.budget
FROM Joueur J, Club C, Finance F
WHERE J.cnum = C.cnum AND  C.cnum = F.cnum
AND C.division = 1 AND J.salaire > 59000 AND J.sport = 'aviron'
```

1.Déterminer un arbre d'opérateurs de l'algèbre relationnel qui reflète l'ordre des opérations qu'un optimiseur de requête peut choisir. Donner la taille l'espace de recherche.

2. Pour réduire l'espace de recherche exploré pendant l'optimisation, on considère seulement les arbres de jointure qui n'ont pas de produit cartésien et qui sont linéaires à gauche. Donner la liste de tous les arbres de jointure construits. Expliquer comment vous obtenez cette liste.

3. Les informations suivantes sont extraites du catalogue du SGBD :Les attributs Joueur.cnum, Joueur.salaire, Club.division, Club.cnum et Finance.cnum sont indexé par un arbre B+ secondaire. Le salaire d'un joueur est compris entre 10.000 et 60.000 EUR. Les joueurs peuvent pratiquer 200 sports différents. Un club est en division 1 ou 2. La BD contient au total 50000 joueurs et 5000 club. Il y a un tuple d'information financière par club. Le seul algorithme de jointure implémenté dans le SGBD est la jointure par boucles imbriquées.

a) Pour chaque relation de la base de données (Joueur, Club et Finance) estimer le nombre de tuples qui sont sélectionnés après avoir traité les prédicats de sélection et avant de traiter les jointures.

b) D'après la réponse à la question précédente, quel est l'arbre de jointure de coût minimum que l'optimiseur construit ?

Exercice 4 : Coût d'une jointure

Soit deux relation R(X,Y) et S(Y,Z). Le nombre de tuples de R et S est T(R) et T(S).On T(S)<T(R)
Le nombre de bloc (ou page) pour stocker une relation est B(R) et B(S).
La taille de la mémoire disponible est de M blocs. M < B(S) < B(R)
Déterminer le nombre N minimal de bloc à lire pour traiter l'equi-jointure de R avec S selon l'algorithme de jointure utilisé :
1) jointure par boucle imbriquée en lisant les relations tuple par tuple.
2) jointure par boucle imbriquée en lisant les relations bloc par bloc
3) jointure par tri fusion.

EXTRAIT DU PARTIEL BDA (AVRIL 2002)

Exercice 5 : Dictionnaire pour les statistiques pts

Un SGBD maintient des statistiques sur les données de la BD. Périodiquement et si les données ont été mises à jour, le SGBD actualise les statistiques au moyen de requêtes SQL.
Soit la relation R(a1, a2, a3). La distribution des valeurs des attributs de R est uniforme.
a) Donner les requêtes SQL pour calculer les statistiques suivantes :
Les bornes du domaine de a1
Le nombre de valeurs distinctes de a2
b) Donner une requête SQL pour vérifier que a2 et a3 sont indépendants. Expliquer comment interpréter le résultat de la requête.
c) Le dictionnaire du SGBD est modifié pour gérer la distribution **non** uniforme des attributs
Relation(nomRel, cardinalité), Attribut(nomRel, nomAtt, val, freq)
Un attribut est identifié par son nom *nomAtt* et le nom de sa relation *nomRel*. ; *val* représente une valeur de l'attribut ; *freq* représente le nombre de tuples dont l'attribut *nomAtt* vaut *val*.
Ecrire, en pseudo-code, un algorithme pour collecter ces statistiques en utilisant des ordres SQL pour manipuler les données du dictionnaire.
Donner en fonction de *R*, *att* et *v*, la requête SQL (sur les relations du dictionnaire) pour déterminer le **facteur de sélectivité** (SF) de l'opérateur de sélection sur la relation *R* avec le prédicat *att*<*v* lorsque la distribution de l'attribut *att* est non uniforme. (*ie*. $SF(\sigma_{att<v}(R))$)

Exercice 6 : Optimisation de requêtes pts

Soit les relations R(a,b), S(c), la distribution des attributs est uniforme,
Le domaine des attributs est : $a \in [1,1000]$, $b \in [1,10]$, $c \in [1,100]$.
La cardinalités des relations sont : card(R) = 1000, card(S)= 100
La taille en nb de pages des relations est : P(R) = 500, P(S) = 10
Le modèle de coût simplifié est :
Le coût d'une jointure '∞' entre A et B, avec le prédicat A.att = B.att est :

coût($A \infty_{A.att=B.att} B$) = card(A) * coût ($\sigma_{att=x}(B)$) où x est une valeur quelconque.

Le coût d'une sélection sur A avec le prédicat att=val est :

si att est indexé : coût($\sigma_{att=val}(A)$) = card(($\sigma_{att=val}(A)$)

sinon : coût($\sigma_{att=val}(A)$) = P(A).

Le coût d'une lecture séquentielle est le coût d'une sélection sans index.
Le coût d'un opérateur traité en pipeline est nul.

Soit la requête : select * from R,S where a=c and b =1
Pour chaque cas a), b) et c), représenter le plan d'exécution de coût minimal de la requête, sous la forme d'un arbre d'opérateurs. Donner le coût et la cardinalité de chaque opérateur de l'arbre, donner le coût total de l'arbre.

a) il y a seulement un index sur R.a et un index sur R.b,

b) il y a seulement un index sur S.c,

c) il y a seulement un index sur R.b et un index sur S.c.

Extrait du partiel BDR (avril 2003)

Exercice 1 : Optimisation — 6 pts

On considère la jointure naturelle des relations R(a,b), S(c,b), T(c,d) et U(a,d).
Pour traiter un arbre de jointures, le SGBD exécute les jointures les unes après les autres, en matérialisant le résultat d'une jointure avant de traiter la jointure suivante.
Le coût d'un arbre de jointures est le nombre de n-uplets matérialisés dans les relations intermédiaires.
On souhaite trouver l'arbre de jointures de coût minimum.

Le modèle de coût d'une jointure est donné par la règle :

coût (A ⋈ B) = card (A ⋈ B) pour un nœud interne (car la relation intermédiaire est matérialisée)
= 0 pour la racine (car le résultat est directement envoyé au processus client)

Les relations R et U ont 1000 n-uplets alors que S et T ont 100 n-uplets.Il y a 100 valeurs différentes pour tous les attributs de toutes les relations excepté pour S.c et T.c qui ont seulement 10 valeurs différentes. On note ces informations de la manière suivante :

card(R) = card(U) = 1000
card(S) = card(T) = 100
nb_val(R.a) = nb_val(R.b) = nb_val(S.b) = nb_val(T.d) = nb_val(U.a) = nb_val(U.d) = 100
nb_val(S.c) = nb_val(T.c) = 10

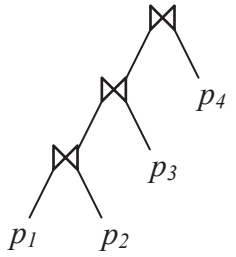
La cardinalité du résultat d'une jointure est donnée par la formule :

card(A ⋈ B) = card(A) * card(B) / max(nb_val(A.*x*), nb_val(B.*x*)) (*x* est l'attribut de jointure)

On rappelle que la jointure naturelle de deux relations est une équi-jointure sur les attributs de mêmes noms.

Question 1 :

On considère uniquement les arbres linéaires gauches de la forme :



1.1) Parmi R, S, T, U, quelles relations peuvent être à la place p_1 ?

1.2) Si R est à la place p_1, quelles relations peuvent être à la place p_2 ?

1.3) Combien existent-ils d'arbres linéaires gauches équivalents pour la jointure naturelle de R, S, T, U ?

1.4) Donner un des arbres linéaires gauches de plus faible coût ? Donner son coût.

Question 2 :

Si on considère maintenant tous les arbres de jointures : les arbres linéaires gauches, les arbres linéaires droits et les arbres équilibrés (dits touffus).

Quel est un des arbres de plus faible coût ? Donner son coût.

Extrait du partiel BDR (avril 2004)

Exercice 3 : Optimisation de requêtes 4 pts

Soit le schéma relationnel décrivant l'organisation d'un laboratoire en départements contenant des employés et des projets.

Emp (Enum, salaire, age, Dnum)

Dept (Dnum, Pnum, budget, statut)

Proj (Pnum, code, description)

Tous les attributs sont des nombres entiers sauf *statut* et *description* qui sont des chaînes de caractères. Les attributs soulignés forment une clé. La clé de Dept est composée de deux attributs, (l'attribut Dnum seul n'est pas une clé primaire de Dept).

La taille d'un n-uplet est respectivement de 20 octets pour Emp, 40 octets pour un Dept et 2000 octets pour Proj. Le nombre de n-uplets par relation et respectivement de 20 000 pour, Emp, 5000 pour Dept et 1000 pour Proj. La taille d'une page, sur le disque ou en mémoire, vaut 4000 octets. La fonction page(R) retourne le nombre de pages contenant les n-uplets de R.

La distribution des valeurs des attributs est uniforme. Les attributs sont indépendants. Le domaine de l'âge des employés est l'ensemble des nombres entiers de 20 à 69 inclus : {20, 21, …, 69}. Le domaine des budgets est l'ensemble des multiples de 1000 inclus dans]100 000, 600 000].

On suppose que tous les index sont des arbres B+. Un index sur l'attribut A est dit *plaçant* si les données sont **triées** sur le disque dans l'ordre des valeurs de A. Un index sur l'attribut A est dit *non plaçant* si les données ne sont **pas triées** sur le disque dans l'ordre de A.

L'estimation du coût des opérations repose sur le modèle suivant:

- Le coût d'une lecture séquentielle de la relation R est égal au nombre de pages de R.
- Le coût d'une sélection avec un prédicat *pred* de la forme *a op v*

  où *a* est un attribut de type entier et *op* est l'opérateur = (égal) , < (inférieur à) ou < (supérieur à)

  coût(σ_{pred} (R)) = card(σ_{pred} (R)) si l'attribut *a* est indexé par un index non plaçant,

  coût(σ_{pred} (R)) = page(R) * card(σ_{pred} (R)) / card(R) si l'attribut *a* est indexé par un index plaçant,

  coût(σ_{pred} (R)) = **page**(R) si l'attribut *a* n'est pas indexé.

Précisez clairement toute hypothèse supplémentaire que vous jugez nécessaire pour répondre aux questions posées.

Question 1. Soit la requête R1: `select * from Emp where age=30`

a) Quel est le coût pour traiter R1 s'il existe un index non plaçant sur l'attribut *age* ?

b) Quel est le coût pour traiter R1 s'il existe un index plaçant sur l'attribut *age* ?

Question 2. Soit la requête R2 `select * from Proj where code=20`

On suppose que la cardinalité de R2 est identique à celle de R1.

a) Quel est le coût pour traiter R2 s'il existe un index non plaçant sur l'attribut *code* ?

b) Quel est le coût pour traiter R2 s'il existe un index plaçant sur l'attribut *code* ?

Question 3. Soient *n* un nombre entier et la requête R3 dépendant de *n* :

```
select * from Dept where budget > n
```

a) On suppose qu'il existe un index **non plaçant** sur l'attribut *budget*. Pour quelles valeurs de *n*, la lecture séquentielle de Dept est moins coûteuse que l'accès par index, pour traiter R3 ? Répondre en donnant toutes les valeurs de *n*.

b) Même question mais avec un index **plaçant** sur l'attribut *budget*. On suppose, pour cette question, que la traversée de l'index coûte 3 lectures.

BDR
EXTRAIT Partiel du 5 avril 2005

Exercice 2 : Optimisation de requêtes 10 pts

Soit le schéma suivant :

Emp (ne, salaire, age, ns) -- un employé est identifié par son numéro *ne*

Service (ns, np, budget, statut) -- *ns* fait référence à un numéro de service.

Projet (np, code, libellé) -- un projet est identifié par son numéro *np*

La taille d'un n-uplet est de 20 octets pour Emp, 40 octets pour Service, 2000 octets pour Projet. Les attributs *ne, ns* et *np* ont chacun 4 octets. La cardinalité des relations est de 20 000 pour Employés, 5000 pour Service et 1000 pour Projet. La relation Service représente l'**association N-M** d'un service avec un projet. La clé de Service est composée des attributs *ns* et *np* (ainsi, l'attribut *ns* n'est pas unique dans Service).

Chaque Service, identifié par *ns* a en moyenne 10 Projets. Les données sont stockées sur disque dans des pages de 4000 octets. Les attributs sont indépendants et leur distribution est uniforme.

Soient les fonctions auxiliaires :

ntp(R) le nombre de tuples par pages pour la relation *R*,

D(R, c) le nombre de valeurs distinctes du domaine de l'attribut *R.c*,

largeur(R) la taille d'un tuple de R,

et *arr(x)* l'arrondi de *x* par excès à une valeur entière,

Un index sur l'attribut *c* est dit *plaçant* si les données sont **triées** sur le disque dans l'ordre des valeurs de *c*. Un index sur l'attribut *c* est dit *non plaçant* si les données ne sont **pas triées** sur le disque dans l'ordre de *c*.

L'estimation du coût des opérations repose sur le modèle suivant. Le modèle de coût estime le nombre de pages à lire et écrire, sans prendre en compte l'écriture du résultat final.

- Le coût d'une lecture séquentielle de la relation R est égal au nombre de pages de R (noté page(R)).

Le coût d'une sélection avec un prédicat *pred* de la forme *a op v* où *a* est un attribut numérique et *op* est l'opérateur = (égal), < (inférieur à) ou < (supérieur à), est :

coût(σ_{pred} (R)) = card(σ_{pred} (R)) si l'attribut *a* est indexé par un index non plaçant,

coût(σ_{pred} (R)) = arr(page(R) * card(σ_{pred} (R)) / card(R)) si l'attribut *a* est indexé par un index plaçant,

coût(σ_{pred} (R)) = page(R) si l'attribut *a* n'est pas indexé.

- Le coût d'une jointure naturelle avec un prédicat de jointure *pred* de la forme *R.c = S.c* dépend de l'algorithme utilisé et de la présence d'index sur les attributs de jointure. Les algorithmes de jointure sont numérotés J1 à J9 :

- J1: jointure par boucles imbriquées, l'itération sur R imbrique l'itération sur S, sans utiliser aucun index.
- J2: jointure avec itération sur *R* et accès par index plaçant sur *S.c*
- J3: jointure avec itération sur *R* et accès par index **non** plaçant sur *S.c*
- J4 : jointure par tri de *R* et *S* selon *c*, puis fusion sans utiliser aucun index
- J5 : jointure par fusion, avec des index plaçants sur *R.c* et sur *S.c*
- J6 : jointure par fusion, avec des index **non** plaçants sur *R.c* et sur *S.c*

- J7 : jointure par tri de *R* selon *c*, puis fusion en utilisant un index plaçant sur *S.c*
- J8 : jointure par tri de *R* selon *c*, puis fusion en utilisant un index **non** plaçant sur *S.c*
- J9 : création d'une table de hachage (non plaçante) de *R* sur *c*, puis jointure avec itération sur *S* et accès à *R* par la table de hachage.

On propose 9 formules de coût :

C1: coût(R $\bowtie_{pred}$ (S)) = page(R) + card(R) x card(S) / D(S, c)

C2: coût(R $\bowtie_{pred}$ (S)) = 3 x page(R) + card(S)

C3 : coût(R $\bowtie_{pred}$ (S)) = 3 x (page(R) + page(S))

C4 : coût(R $\bowtie_{pred}$ (S)) = page(R) + page(S)

C5: coût(R $\bowtie_{pred}$ (S)) = page(R) + card(R) x page(S)

C6: coût(R $\bowtie_{pred}$ (S)) = page(R) + card(R) x arr(page(S) / D(S, c))

C7: coût(R $\bowtie_{pred}$ (S)) = page(R) + page(S) + card(S) x card(R) / D(R,c)

C8: coût(R $\bowtie_{pred}$ (S)) = 3 x page(R) + page(S)

C9: coût(R $\bowtie_{pred}$ (S)) = card(R) + card(S)

Précisez clairement toute hypothèse supplémentaire que vous jugez nécessaire.

Question 1

1) Donner le nombre de services distincts et le nombre moyen de services par projet

2) Donner le nombre de tuples par page des 3 relations.

3) Donner la taille des relations en nombre de pages.

4) Combien de pages sont nécessaires pour stocker conjointement tous les tuples de R qui ont une même valeur pour l'attribut *c* ? Donner une formule en fonction de *R*, *c* et des notations définies ci-dessus.

Question 2

1) Parmi C1 à C9, quelles sont les formules symétriques (*i.e.*, pour lesquelles R et S ont le même rôle)
2) Parmi J1 à J9, quels sont les algorithmes symétriques (*i.e.*, pour lesquels R et S ont le même rôle)
3) Associer chaque algorithme de jointure avec la formule la plus appropriée. Expliquer vos choix en une phrase. Répondre en complétant le tableau. Indication :

Coût	Explication
J1 : C5	page(R) : lire R une seule fois card(R) * page(S): lire S autant de fois qu'il y a de tuples dans R

Question 3.

Soit la requête R1:

```
Select *
From Emp e, Service s
Where e.ns = s.ns
```

1) On suppose qu'il existe seulement un index plaçant sur Emp.ns. Dire, pour chaque algorithme J1 à J9, s'il est utilisable pour traiter R1? Si oui, donner le coût du plan d'exécution de R1. Sinon expliquer brièvement pourquoi.

2) On suppose maintenant qu'il existe un index plaçant sur Emp.ns et un index plaçant sur Service.ns.

Quel est l'algorithme de coût minimal ? Donner son coût.

3) On suppose maintenant qu'il existe seulement un index plaçant sur Service.ns. Le SGBD possédant seulement 7 pages disponibles en mémoire, on choisit de traiter R1 avec l'algorithme suivant :

Etape 1: Lire Emp par blocs de 6 pages (1 page mémoire étant réservée pour écrire le résultat du tri), et créer des sous-listes triées.

Etape 2: Tant que le nombre de sous-listes est supérieur ou égal à 6 :

fusionner 6 sous-listes en 1 seule liste, écrire le résultat sur disque.

Etape 3: Fusionner les listes restantes avec la relation Service

a) Donner le coût de R1, en utilisant cet algorithme, en nombre de pages (lues ou écrites).

b) Combien faut-il de pages mémoire au minimum pour pouvoir utiliser J7 ?

Question 4.

Rappel des formules évaluant la cardinalité de la sélection et de la jointure :

$card(\sigma_F (R)) = SF(\sigma_{(F)}) * card(R)$

où $SF(\sigma_{A = valeur}) = 1 / D(R, A)$

$SF(\sigma_{A > valeur}) = (max(A) - valeur) / (max(A) - min(A))$

$SF(\sigma_{A < valeur}) = (valeur - min(A)) / (max(A) - min(A))$

$card (R \bowtie_{A=B} S) = card(S)$ si A est clé de R, et B est clé étrangère de S.

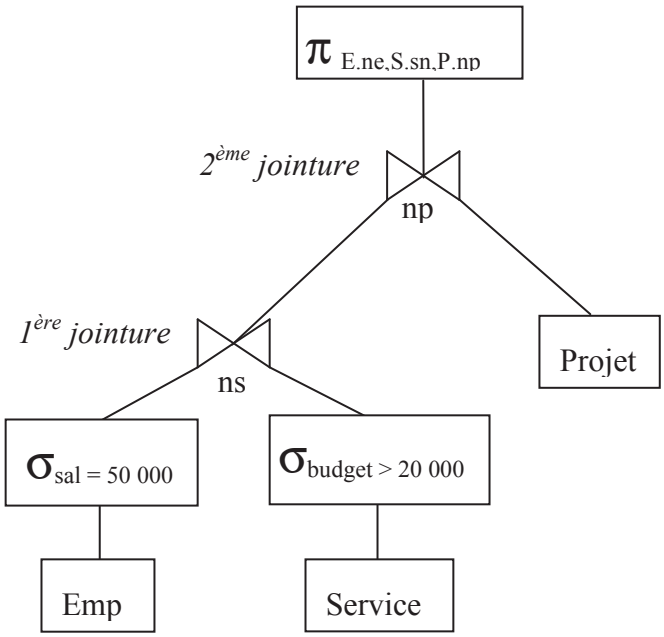
sinon $card (R \bowtie_{A=B} S) = SFj * card(S) * card(R)$ où SFj = 1 / max(D(R, A), D(S, B))

On donne SFj = 1/500 pour la jointure entre Emp et Serv.

Soit la requête **R2** :

```
Select E.ne, S.ns, P.np
From Emp E, Service S, Projet P
Where E.Salaire = 50000
      and S.budget > 20000
      and E.ns=S.ns and S.np=P.np
```

On suppose une répartition uniforme des salaires (par tranche de 5000 dans l'intervalle [10000, 105000]) et des budgets (dans l'intervalle [10000, 30000]). Il y a un index plaçant sur l'attribut salaire de Emp, sur l'attribut ns de Service et sur l'attribut np de Projet.
On considère le plan d'exécution **P1** suivant :



1) Estimez le nombre de n-uplets résultant de chacune des opérations.

card ($\sigma_{sal = 50\,000}$ (Emp)) = ...

card ($\sigma_{budget > 20\,000}$ (Service)) = ...

card (($\sigma_{sal = 50\,000}$ (Emp)) $\bowtie_{ns}$ ($\sigma_{budget > 20\,000}$ (Service))) = ...

card (résultat final) =

2) Calculez le coût du plan d'exécution P1, en précisant les algorithmes de jointure utilisés.

coût ($\sigma_{sal = 50\,000}$ (Emp)) = . . .

coût ($\sigma_{budget > 20\,000}$ (Service)) = . . .

coût (1ère jointure) = ...

coût (2ème jointure) = ...

coût total = ...

3) Donnez un plan d'exécution de coût optimal pour la requête R2 et calculez son coût.

4) On suppose maintenant que l'index sur l'attribut *np* de Projet est non plaçant. Calculez le coût du plan optimal sous cette hypothèse.
5) Même question en supposant que les index sur *salaire* et sur l'attribut *ns* de *Service* sont non-plaçants. Justifiez votre réponse.

TD - Conception de BD réparties - Requêtes

Le but de ce TD est l'étude de la conception d'une base de données répartie. Dans la première partie, une base dont le schéma conceptuel a été défini est distribuée sur plusieurs sites. Le but principal de la distribution est de maximiser les accès locaux par rapport aux accès répartis. Dans la seconde partie, une base préexistante est intégrée au système. Dans ce cas, l'intégration doit être réalisée sans modification de la base locale qui possède ses propres applications locales déjà écrites, sous la forme d'une vue répartie. La troisième partie est une étude quantitative du coût de traitements des requêtes sur une BD répartie. Enfin, la dernière partie est une étude de cas.

1 - Conception de BD réparties

Schéma global de la base

La base de données hospitalière de la région Alsace a le schéma suivant :

Service (Snum, nom, hôpital, bât, directeur)

le directeur d'un service est un docteur désigné par son numéro

Salle (Snum, SAnum, surveillant, nbLits)

le numéro de salle est local à un service, i.e., il peut y avoir des salles avec le même numéro dans des services différents d'un même hôpital.

nbLits est le nombre total de lits d'une salle,

un surveillant de salle est un infirmier désigné par son numéro

Employé (Enum, nom, adr, tél)

Docteur (Dnum, spéc) -- *spéc est la spécialité du médecin*

Infirmier (Inum, Snum, rotation, salaire)

Un employé est soit infirmier soit docteur (Inum et Dnum font référence à Enum).

Patient (Pnum, Snum, SAnum, lit, nom, adr, tél, mutuelle, pc)

L'attribut pc est la prise en charge par la mutuelle

Acte (Dnum, Pnum, date, description, coef) -- *coef est le coefficient de l'acte médical*

Question 1

Exprimer en SQL la question suivante: "Donner le nom des cardiologues qui ont traité un ou plusieurs patients hospitalisés dans un service de gérontologie."

Répartition des données

La base est répartie sur trois sites informatiques, "Strasbourg", "Colmar" et "Régional", correspondant aux valeurs "*Ambroise Paré*", "*Colmar*" et "*autre"* de l'attribut *hôpital* de Service.

Question 2

Proposer (et justifier) une bonne décomposition de la base hospitalière sur ces trois sites. On pourra utiliser la fragmentation horizontale et/ou verticale ainsi que la réplication des données, en se basant sur les hypothèses suivantes (H1 à H5) :

• H1: Les sites Strasbourg et Colmar ne gèrent que les hôpitaux correspondants.

• H2 : Les infirmiers sont employés dans un service donné.

• H3 : Les docteurs travaillent le plus souvent sur plusieurs hôpitaux (ou cliniques).

• H4 : La gestion des lits d'hôpitaux est locale à chaque hôpital.

• H5 : On désire regrouper la gestion des frais d'hospitalisation au centre régional.

Pour chaque fragment, on donnera sa définition en algèbre relationnelle à partir du schéma global.

Question 3

Indiquer comment se calcule chaque relation de la base globale à partir de ses fragments.

Question 4

Proposer un plan d'exécution réparti pour la requête SQL vue en Question 1, sachant maintenant que les données sont réparties sur les trois sites selon la décomposition proposée à la Question 2.

2 - Conception de BD fédérées par intégration

On suppose que l'hôpital de Belfort est rattaché à la base de donnée hospitalière de la région Alsace après son implémentation répartie. L'hôpital de Belfort possède donc son propre site de traitement qui doit être connecté aux autres sites.

Le schéma de la base à Belfort avant l'intégration est le suivant:

B_Service (Snum, Nom, Bâtiment, Directeur)

B_Salle (Snum, SAnum, Surveillant, NbLits)

B_Docteur (Dnum, Nom, Adresse, Téléphone, spécialité)

B_Infirmier (Inum, Nom, Adresse, Téléphone, Snum, Salaire)

B_Patient (Pnum, Snum, SAnum, Lit, Nom, Adresse, Téléphone, Mutuelle, PriseEnCharge)

B_Acte (Dnum, Pnum, Date, Description, Code)

Question 5

Discuter les problèmes et proposer des solutions pour l'intégration de la base Belfort au système réparti déjà défini. L'intégration devra se faire sans transfert d'information et sans modification de des bases existantes, mais uniquement par définition de vues.

Question 6

Définir le nouveau schéma global intégrant la base Belfort. Chaque relation du schéma global (Service2, Salle2, …Acte2) est définie en fonction des fragments sur les 4 sites.

Question 7

L'hypothèse H5 est-t-elle toujours respectée après l'intégration de la base Belfort ? Si non, quelles sont les modifications de schéma nécessaires pour respecter H5 ?

Question 8

Proposer une décomposition et un plan d'exécution pour la question SQL précédente après l'intégration de la base "Belfort".

3 - Evaluation de requêtes réparties (juin 2002)

On considère la base de données répartie de schéma suivant **:**

Employés (#emp, #serv, salaire)

Service (#serv, #dir, budget)

L'attribut #dir représente l'identificateur de l'employé dirigeant le service.

La relation Employés est stockée à Naples, la relation Service est stockée à Berlin.

La taille des n-uplets de ces deux relations est de 20 octets. La taille de **#emp** et de **#dir** est de 10 octets. Les attributs **salaire** et **budget** contiennent des valeurs uniformément réparties dans l'intervalle [0, 1 000 000]. La relation **Employés** comprend 100 000 pages, la relation **Service** comprend 5000 pages. Chaque processeur a **400** pages de buffer. La taille d'une page (du buffer et du disque) est de 4000 octets. Le coût d'entrée/sortie d'une page est t_{IO}, le coût de transfert d'une page d'un site à un autre est t_T. L'unité de transfert est la page. On suppose qu'il n'y a pas d'index.

On considère la requête suivante :

```
SELECT *
FROM Employés E, Service S
WHERE E.#emp = S.#dir
```

La requête est envoyée de Londres, et on sait que 1% des employés sont des directeurs.

Question 1.

a) Calculer la taille d'un tuple du résultat.

b) Calculer la cardinalité du résultat.

c) Calculer la taille (en nombre de pages) du résultat de cette requête.

Question 2.

On suppose que toutes les jointures sont faites en utilisant l'algorithme de tri-fusion, dont le coût dépend du nombre de pages (noté *p()*) des relations participant à la jointure. On a :

$Coût(R \bowtie_a S) = Tri_a(R) + Tri_a(S) + [Lect(R) + Lect(S)]$

Tri(R) = Lect(R) + Ecr(R) si les données ne sont pas déjà triées sur les attributs de jointure
Lect(R) = p(R) si les données sont locales
= 0 si les données sont transmises directement depuis un site distant
Ecr(R) = p(R) // correspond au stockage temporaire des données

Par exemple, pour une jointure entre 2 relations locales non triées on a :

$Coût(R \bowtie_a S) = 3 * (p(R) + p(S)) * t_{IO}$

Pour une jointure entre 2 relations locales triées, on a :

$Coût(R \bowtie_a S) = (p(R) + p(S)) * t_{IO}$

Si R est déjà trié et est transmise depuis un site distant et si S est locale non triée, on a

$Coût(R \bowtie_a S) = (3*p(S)) * t_{IO}$

On demande de calculer le coût de l'exécution de cette requête pour les différents plans d'exécution suivants :

Plan P1: On calcule la requête à Naples en envoyant la relation Service à Naples ; le résultat est envoyé à Londres.

Plan P2: On calcule la requête à Berlin en envoyant la relation Employés à Berlin ; le résultat est envoyé à Londres

Plan P3: On calcule la requête à Londres, en envoyant les deux relations à Londres.

Plan P4: On calcule la requête à Berlin puis à Naples, en utilisant une semi-jointure à Berlin ; on envoie le résultat final à Londres. (attention, la semi-jointure peut amener à créer des relations temporaires qu'il faudra intégrer dans le coût)

Plan P5: On calcule la requête à Naples puis à Berlin en utilisant une semi-jointure à Naples ; on envoie le résultat à Londres.

Question 3.

D'après vos réponses, quel est le plan qui minimise les transferts ? Est-ce forcément le plan le plus intéressant ? Quel est le meilleur plan ?

4 - Fragmentation et requête répartie (bdweb janvier 2002)

La base de données d'un revendeur en ligne, PointCom, a le schéma global suivant :

PRODUITS (Num-Produit, Catégorie, Titre, Description, Prix, …)

INVENTAIRE (Num-Produit, Entrepôt, Nbr-Exemplaire, …)

COMMANDES (Num-Commande, Num-Produit, Adresse-Livraison, Quantité, …)

PointCom offre quatre catégories de produits : livres, musique, vidéos, et jeux, et possède trois entrepôts situés en Californie (CA), New York (NY) et Colorado (CO).

- L'entrepôt de la Californie gère des livres, de la musique, et des vidéos, et effectue des livraisons pour des commandes vers les régions de l'ouest et du centre des États-Unis.
- L'entrepôt de New York gère des livres, des jeux, et des vidéos, et effectue des livraisons pour des commandes vers les régions de l'est et du centre des États-Unis.
- L'entrepôt de Colorado gère des livres, des jeux, et de la musique, et effectue des livraisons pour des commandes vers toutes les régions des États-Unis.
- Num-Produit est la clé primaire de PRODUITS.
- Catégorie prend une des valeurs suivantes : "livre", "musique", "vidéo", et "jeux".
- Entrepôt prend une des valeurs suivantes : "CA", "NY", et "CO".
- Adresse-Livraison prend une des valeurs suivantes : "Est", "Ouest", et "Centre".

Question 1

Exprimer en SQL sur le schéma global la requête permettant de retrouver le nombre d'exemplaires disponibles du livre intitulé " X " pour une livraison vers l'ouest des États-Unis.

Question 2

Supposons maintenant que la base PointCom est répartie sur les trois sites informatiques de la Californie, de New York, et du Colorado. Proposer une bonne décomposition de la base sur ces trois sites. Donner, en algèbre relationnel, la définition des différents fragments.

Question 3

Proposer un plan d'exécution répartie pour la requête de la question 3 minimisant les transferts entre les sites (vous décidez du site qui pose initialement la requête).

TD : Conception des BD Réparties

Exercice 1 : Fragmentation (juin 2004) pts

Le schéma global d'une base de données cinématographique est le suivant :

Film (F, titre, année, durée, réalisateur).	un film est identifié par son numéro *F*
Artiste (A, nom, prénom, nationalité)	un artiste est identifié par son numéro *A*
Rôle (nom, F, A)	l'artiste numéro *A* joue le rôle *Nom* dans le film numéro *F*.
Cinéma (C, arrond, ville)	un cinéma est identifié par son nom *C*.
Salle (C, Sa, P, clim)	la salle numéro *Sa* du cinéma numéro *C* a une capacité de *P* places. L'attribut *clim*, valant *'oui'* ou *'non'*, indique si la salle est climatisée.
Séance (C, Sa, H, F)	le film numéro *F* est projeté dans la salle numéro *Sa* du cinéma *C* à partir de l'heure *H*.

Les attributs soulignés forment la clé d'une relation.

Question 1: Un spectateur Si possède une partie de la base sur son site personnel. Un spectateur *cinéphile* ne s'intéresse qu'aux films anciens réalisés jusqu'à 2000 inclus, un spectateur *tendance* ne s'intéresse qu'aux films récents réalisés en 2001 et après. Un spectateur ne s'intéresse qu'à des films projetés dans sa *ville* : Paris, Lyon ou Nice. Un spectateur s'intéresse à toutes les données liées aux films qui l'intéressent. Par contre, un spectateur ne s'intéresse jamais à la capacité d'une salle.

Remarque : On peut déduire de l'énoncé qu'un spectateur *cinéphile* ne s'intéresse pas aux films récents dans lesquels jouent des acteurs qui jouent aussi dans des films anciens.

Définir une fragmentation de la base cinématographique qui permette à un spectateur de construire sa base personnelle en répliquant des fragments sur son site, en respectant les conditions suivantes :

- un fragment est répliqué soit entièrement, soit pas du tout, sur le site d'un spectateur,

- la base d'un spectateur doit contenir toutes les données qui l'intéressent, et seulement celles-ci.

Pour **chaque relation** du schéma global :

a) Les n fragments d'une relation R sont notés R_i avec $i \in [1, n]$. Donner le nombre de fragments et écrire les expressions algébriques de définition des fragments.

b) La fragmentation proposée est-elle complète ? Est elle disjointe ? Si non expliquez pourquoi.

c) Définir le schéma de reconstruction de la relation à partir des ses fragments. Donner son expression algébrique

Exercice 2 : Optimisation de requêtes réparties (juin 2004) pts

Soit le schéma global

Artiste (A, nom, prénom, nation) un artiste est identifié par son numéro *A*

Rôle (nom_rôle, F, A) l'artiste numéro *A* joue le rôle *nom_rôle* dans le film numéro *F*.

Les données sont fragmentées de la manière suivante :

Artiste$_i$ = σ p_i (Artiste) où p_i est de la forme *nation = 'n$_i$'* avec $n_i \in$ {D, F, US}

Rôle$_i$ = (Rôle $\ltimes_A$ Artiste$_i$)

Les données sont réparties sur 4 sites dans 3 pays. Chaque pays contient les données en lien avec les artistes citoyens du pays :

Berlin (**B**) contient Artiste$_1$ et Rôle$_1$.

Paris (**P**) contient Artiste$_2$ et Rôle$_2$.

New York (**NY**) et Los Angeles (**LA**) contiennent les mêmes données : Artiste$_3$ et Rôle$_3$.

Les valeurs de tous les attributs, **sauf nationalité**, sont uniformément distribuées. Tous les attributs sont indépendants. Il y a 20 rôles par acteur, 10 rôles par film
Il y a 100 artistes allemands, 100 artistes français et 1000 artistes américains. On considère que tous les artistes sont acteurs. Un acteur n'a jamais 2 rôles dans le même film.

Il n'y a pas d'homonyme : card(π_{nom} Artiste) = card(Artiste)

Le coût d'un traitement local sur un site est négligeable par rapport au coût d'un transfert intersite.
Le coût de transfert dépend de la distance entre les sites. La fonction *tr(T, X, Y)* donne le coût pour transférer le résultat de l'expression algébrique *T* depuis le site *X* vers le site *Y*. On a:

tr(T, NY, P) = card(T)
tr(T, LA, NY) = 2* card(T)
tr(T, LA , P) = 3* card(T)
tr(T, X, Y) = tr(T, Y, X)

Le coût d'un plan d'exécution est la somme de tous les transferts nécessaires pour exécuter le plan.

Question 1.

a) Quelles sont les cardinalités de Artiste, Rôle, π_F(Rôle) $\sigma_{nation = 'US'}$ Artiste ?

b) Que vaut le facteur de sélectivité SF($\sigma_{nation = 'US'}$ Artiste) ?

Question 2.

On souhaite traiter les requêtes suivantes. Pour chaque requête, donner le plan d'exécution de **coût minimal**, en précisant l'expression algébrique de chaque sous requête T_i locale et les transferts. Utiliser la notation *tr(T_i, X, Y)* pour désigner les transferts. Le site sur lequel la requête est posée, reçoit le résultat.

a) Requête **R1** posée à Los Angeles (LA) : Donner le nom des acteurs français qui ont joué avec l'acteur américain nommé Pitt (*i.e.*, dans le même film que lui).

select a2.nom
from Artiste a1, Artiste a2, Role r1, Role r2
where a1.nom = 'Pitt' **and** a1.nationalité= 'US' **and** a2.nationalité = 'F'
and a1.a = r1.a **and** r1.f = r2.f **and** r2.a = a2.a

```
Etapes du plan de coût minimal :
1) sur le site ...
       traiter T1 = ...
       tr(T1, ... , ...) =

2) sur le site ...
       traiter T2 = ...
       tr(T2, ... , ...) =

etc...                  →                                    coût total = ..
```

b) La requête **R2** est posée à Paris (P) : Quels sont les films dont le casting est franco-américain ?

Plus précisément : Donner le numéro des films dans lesquels jouent au moins un acteur français et au moins un acteur américain. Quel est le plan optimal pour R2 ?

c) On suppose maintenant que les attributs *nation* et *F* ne sont plus indépendants. Pour cela on donne

card (π_F Rôle$_1$) = card (π_F Rôle$_2$) = 1000 et card (π_F Rôle$_3$) = 1200

et 5% des films où figurent au moins un acteur français a également un acteur américain

Quel est le plan optimal pour la requête R2 de la question précédente ?

Exercice 3 : Requêtes réparties (juin 2004) 3 pts

On considère la relation

Employé (E, nom, D, salaire)

Un n-uplet représente un employé identifié par son numéro *E*. Le numéro *D* fait référence au directeur de l'employé. Le directeur est aussi un employé.

Soit la requête R1 :

```
select a.nom
from Employé a, Employé b
where a.D = b.E
and a.salaire > b.salaire
```

Question 1. Traduire la requête R1 en une phrase, en français :

Question 2. Donner l'expression algébrique de la requête R1 en fonction de la relation globale *Employé*, et telle que les opérations les plus réductrices sont traitées le plus tôt possible.

Question 3. La relation *Employé* est fragmentée en 2 fragments *E1* et *E2* tels que:

E1 contient tous les employés dont le salaire est inférieur ou égal à 1000,

E2 contient tous les employés dont le salaire est supérieur à 1000.

Donner l'expression algébrique de la requête *R1*, en fonction des fragments *E1* et *E2*, et telle que l'expression soit de la forme :

$$R1 \Leftrightarrow \pi_{\text{nom}} (\text{T1} \cup \text{T2} \cup \ldots \cup \text{Tn})$$

où les sous expressions *Ti* ne contiennent pas d'union et peuvent ne pas être vides,

les opérations les plus réductrices sont traitées le plus tôt possible.

Préciser combien il y a de sous expressions Ti.

BD Réparties : optimisation de requêtes 6 pts

On considère le schéma relationnel suivant :

```
EMPLOYES (#emp, #service, salaire)
SERVICES (#service, #chef, budget)
```

Les valeurs de l'attribut `#chef` sont des valeurs de `#emp`. La relation `EMPLOYES` contient 100000 pages, la relation `SERVICES` contient 5000 pages.Les n-uplets des deux relations ont 20 octets. Les valeurs des attributs salaire et budget contiennent des valeurs uniformément réparties entre 0 et 1000000. La taille des pages est de 4000 octets.

Les relations sont réparties sur 10 sites, de la façon suivante : la relation SERVICES est partitionnée horizontalement sur les 10 sites par #service, avec le même nombre de n-uplets sur chaque site. La relation EMPLOYES est partitionnée horizontalement en fonction du salaire. Les n-uplets dont le salaire est <= 100000 sont stockés sur le site 1, ceux dont le salaire est compris entre 100000 et 200000 sont sur le site 2, etc. La partition des n-uplets dont le salaire est inférieur à 100000 est fréquemment lue, et très peu souvent mise à jour. Elle est donc répliquée sur tous les sites. Aucune autre partition de la relation EMPLOYE n'est dupliquée.

Décrivez le plan d'exécution des requêtes suivantes, et donnez leur coût, sachant que le coût de transfert d'une page est Ct, et le coût d'un accès disque est Cd.

Question 1. Calculer la jointure naturelle entre EMPLOYES et SERVICES, en utilisant la stratégie qui consiste à envoyer tous les fragments de la plus petite relation vers les sites contenant les n-uplets de la plus grande relation. L'algorithme de jointure utilisé est le trifusion, dont le coût est de 3(M+N)Cd, M et N étant les tailles des relations (N et M sont exprimées ici en nombre de **nuplets**) participant à la jointure.

Question 2. Quel est l'employé le mieux payé ? (en cas de dupliqués, la requête doit renvoyer tous les n-uplets répondant au critère).

Question 3.Quel est le chef le mieux payé ?

BDR - Examen juin 2006

Exercice 1 : Evaluation de requêtes (BD parallèles et BD réparties) 5 pts

On considère un SGBD parallèle dans lequel chaque relation est stockée par partitionnement horizontal des n-uplets sur tous les disques. On a la relation suivante :

Joueurs (*joueur-id* : integer**,** *eq-id* : integer**,** *salaire* : real**)**

On suppose que l'attribut salaire contient des valeurs dans l'intervalle [0, 1 500 000] uniformément distribuées. La relation a 150 000 pages. Une page a une taille de 4K octets, et comprend 200 n-uplets (un n-uplet a une taille de 20 octets). Le coût de lecture d'une page du disque (ou d'écriture sur le disque) est t_d, et le coût d'envoi d'une page d'un processeur vers un autre est t_s. On suppose que la relation a été partitionnée suivant l'algorithme round-robin, et qu'il y a 15 processeurs.

Question 1. Décrivez brièvement le plan d'évaluation de la requête R1, et calculez son coût en termes de temps d'accès (t_d) et temps d'envoi (t_s). On donnera le coût total de la requête, ainsi que le coût en temps écoulé (si plusieurs opérations sont effectuées en parallèle, le temps écoulé est le maximum du temps pris par chacun des processeurs pour faire son travail).

R1 . Quel est le joueur le mieux payé ?

Question 2. La relation Joueurs est maintenant stockée dans un SGBD réparti comprenant 15 sites. Elle est partitionnée horizontalement sur les 15 sites, selon les valeurs de l'attribut *salaire,* en stockant les n-uplets vérifiant la condition *salaire* <=100000 sur le premier site, les n-uplets vérifiant la condition 100000 <*salaire*<=200000 sur le second site, et ainsi de suite. La base est complétée par la relation **Equipes** suivante :

Equipes (*eq-id* : integer, *cap-id* : integer, *pays*: string)

Le champ *cap-id* de la relation Equipes est le *joueur-id* du capitaine de l'équipe. La relation Equipes comprend 4500 pages, et est répartie horizontalement selon l'attribut eq-id. On suppose que la répartition est uniforme (on a le même nombre de n-uplets sur chaque site), et aléatoire (il n'y a pas de critère pour répartir les n-uplets sur les sites). Les n-uplets de la relation Equipes ont une taille de 20 octets.

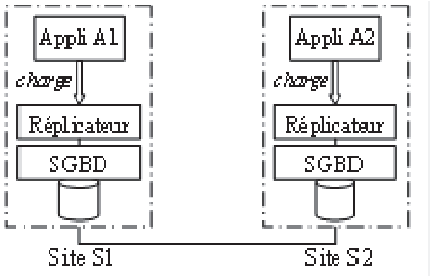
Décrivez brièvement les plans d'exécution des requêtes R1 et R2, et donnez leur coût en fonction de t_d et t_s.

R1. Quel est le joueur le mieux payé ?

R2. Faire la jointure naturelle des relations Joueurs et Equipes en utilisant la stratégie qui consiste à envoyer tous les fragments de la plus petite relation sur chaque site contenant les n-uplets de la plus grande relation.

Exercice 2 : Allocation de fragments 5 pts

On considère l'architecture répartie suivante :



Chaque site héberge un SGBD et une application. Une application produit une charge constituée de lectures et d'écritures. Pour maintenir les sites à jour, on ajoute un module de gestion des répliques (intergiciel appelé réplicateur). Si une donnée est répliquée sur plusieurs sites, une écriture doit être traitée sur **toutes** les répliques. Une lecture est traitée localement si possible.

Soit un fragment F et deux sites S1, S2. Le coût pour lire (resp. écrire) un nuplet vaut 1 (resp. 2). Le coût pour transférer un nuplet vaut 3. Ainsi, on a le modèle de coût suivant :

Lecture locale : LL = 1
Ecriture locale : EL = 2
Lecture distante, depuis Si, d'un nuplet de Sj (avec i≠j) : LD = 4
Ecriture distante, depuis Si, d'un nuplet de Sj (avec i≠j) : ED = 5

On souhaite allouer F sur un ou plusieurs sites de manière à minimiser le coût de traitement des applications.

Question 1
Une application A_i produit, sur le site S_i , une charge valant *6*n* lectures et *n* écritures de F (toutes les applications produisent la même charge). On veut calculer, en fonction de *n*, la quantité de lectures et d'écritures traitées par les SGBD des sites S1 et S2.
1.1) On suppose que F est seulement sur S1. Quel est le coût des traitements sur chaque site ?
1.2) On suppose maintenant que F est répliqué sur S1 et S2. Quel est le coût des traitements sur chaque site ?

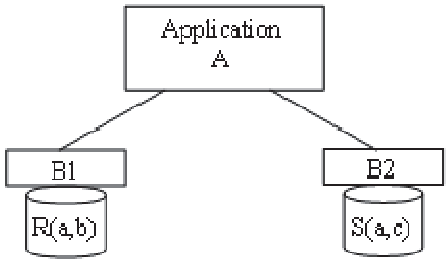
Question 2
On complète l'architecture initiale avec un troisième site S3 et son application A3. Les applications produisent la charge suivante : A2 produit 3 fois plus de traitements que A1, A3 produit 4 fois plus de traitements que A1. Chaque application produit 10 fois plus de lectures que d'écritures. A1 produit *n* écritures et *10*n* lectures.

On veut savoir s'il est intéressant d'allouer F sur S3 ; plus précisément, dans le cas où F ne serait pas alloué sur S3, quelle serait l'augmentation (ou la diminution) du coût que cela induirait pour les traitements de A3.

2.1.a) Quelle est, en fonction des 3 variables n, L (lecture) et E (écritures) la charge produite par chaque application.
2.1.b) Est-il nécessaire d'allouer F sur S3 pour minimiser globalement le coût des traitements ? Justifier brièvement.
2.1.c) Le fait d'allouer F sur S3 a-t-il un impact sur le coût de traitement des lectures de A3 ? Si oui, quelle est la différence de coût pour traiter les lectures de A3 en comparaison avec une configuration où F n'est pas sur S3 ? Donner une réponse en fonction de n.
2.1.d) Quelle est l'allocation pour laquelle F n'est pas sur S3 et les écritures sont minimisées ? Dans ce cas, quel est en fonction de n, le coût de traitement des écritures produites par A3 ?
2.1.e) Quelle est l'allocation pour laquelle F est sur S3 et les écritures sont maximisées ? Dans ce cas, quel est en fonction de n, le coût de traitement des écritures produites par A3 ?
2.1.f) Finalement, quelle est, en fonction de n, la plus petite augmentation (ou la plus forte diminution) de coût apportée par une configuration où F n'est pas sur S3.
2.2) Quel est le coût des traitements si F est seulement sur S3 ?
2.3) Quel est le coût des traitements si F est répliquée sur S2 et S3 ?
2.4) Quel est le coût des traitements si F est répliquée sur S1, S2 et S3 ?
2.5) Quelle allocation minimise globalement le coût des traitements ?

Exercice 3 : JDBC — 5 pts

Les relations **R**(<u>a, b</u>) et **S**(<u>a, c</u>) sont réparties selon l'architecture suivante :



L'application A, écrite en java, communique avec les SGBD B1 et B2 par JDBC. B1 et B2 ne peuvent pas communiquer directement entre eux. Soient les informations suivantes :

card(R) = 1000, card(S)=300, la taille des attributs en octets est a=4, b=10, c=10.
Les valeurs de a, b, c sont des entiers positifs. Il y a exactement 2 nuplets de R par valeur de *a* et 3 nuplets de S par valeur de *a*. R et S ont 4 valeurs distinctes de *a* en commun.

Soient CE le coût unitaire d'exécution d'une requête sur un site et CT le coût unitaire de transfert d'une donnée sur le réseau.

Le coût d'instruction JDBC est le suivant : (toutes les tailles sont en octets)

Interface	**Méthode**	**Coût (en octets)**
Statement	executeQuery(requête)	CE * taille du résultat de la requête
ResultSet	getString(...)	**CT** * taille de l'attribut lu

Toutes les autres instructions ont un coût nul.
Remarques : Le coût correspond au traitement d'une seule occurrence d'une instruction. Le coût total de plusieurs occurrences d'une même instruction, pendant tout le déroulement du programme, est déduit du tableau ci-dessus.

Question 1

Donner la valeur de card($R \bowtie_a S$)

Question 2

L'application exécute le programme P1 ci-dessous :

```
1       Connection c1 = DriverManager.getConnection("B1");
2       Connection c2 = DriverManager.getConnection("B2");
3       Statement s1 =  c1.createStatement();
4       Statement s2 =  c2.createStatement();
5       ResultSet r1 =  s1.executeQuery("select a,b from R");
6       while (r1.next()) {
7         String v = r1.getString(1);
8         ResultSet r2 = s2.executeQuery("select c from S where a = '" + v + "'");
9         boolean p = true;
10        String w = "";
11        while (r2.next()) {
12          if(p) {
13              w = r1.getString(2);
14              p = false;
15          }
16          out.println("[" v + ", " + w + ", " + r2.getString(1) + "]");
17        }
18        r2.close();
19      }
20      r1.close();
21      s1.close(); s2.close(); c1.close();  c2.close();
```

a) Pourquoi le programme contient-il l'instruction conditionnelle *if(p)* en ligne 12 ?
b) Combien de fois, pendant toute la durée d'exécution du programme, l'instruction de la ligne 13 est-elle traitée ?
c) Combien de fois, pendant toute la durée d'exécution du programme, l'instruction de la ligne 8 renvoie-t-elle un résultat vide (dont la taille est nulle) ?
d) Quel est le coût de P1? Pour chaque instruction dont le coût n'est pas nul : donner le n° de ligne, le nombre de fois que l'instruction est exécutée et son coût en fonction de CE et CT. Remarque : le coût d'une ligne correspond au coût **total** de traitement de cette ligne pendant toute l'exécution du programme.

Question 3

L'application exécute le programme P2 ci-dessous:

```
1     Connection c1 = DriverManager.getConnection("B1");
2     Connection c2 = DriverManager.getConnection("B2");
3     Connection c3 = DriverManager.getConnection("B1");
4     Statement s1 = c1.createStatement();
5     Statement s2 = c2.createStatement();
6     Statement s3 = c3.createStatement();
7     ResultSet r1 = s1.executeQuery("select distinct a from R");
8     while(r1.next()) {
9       String z = r1.getString(1);
10      ResultSet r2 = s2.executeQuery("select c from S where a = '" + z + "'");
11      while(r2.next()) {
12        String t = r2.getString(1);
```

```
13        ResultSet r3 = s3.executeQuery("select b from R where a = '" + z + "'");
14        while (r3.next() ) {
15          out.println( "[" + z + ", " + r3.getString(1) + ", " + t + "]");
16        }
17        r3.close();
18      }
19      r2.close();
20    }
21    r1.close();
22    s1.close(); s2.close(); s3.close(); c1.close(); c2.close(); c3.close();
```

P1 et P2 produisent-ils le même résultat? Quel est le coût de P2 ?

Question 4

On considère la requête R1 = R $\bowtie_a$ S. On veut calculer R1 en traitant la jointure par un algorithme de tri fusion. On donne une liste d'instructions parmi lesquelles certaines participent au calcul de R1. Pour chaque instruction utile, donner son numéro, le nombre de fois qu'elle est traitée et son coût. Répondre en triant les instructions dans l'ordre où elles sont traitées pour la première fois.

n°1 : r = executeQuery("select a, b from R order by a");
n°2 : r = executeQuery("select a, b from R order by b");
n°3 : r = executeQuery("select distinct a from R order by a");
n°4 : r.getString(1);
n°5 : r.getString(2);
n°6 : s = executeQuery("select a, c from S order by a");
n°7 : s = executeQuery("select a, c from S order by c");
n°8 : s = executeQuery("select distinct a from S order by a");
n°9 : s.getString(1);
n°10 : s.getString(2);

Exercice 4 : Requêtes réparties 5 pts.

Soit **R1**(a, b) sur le site S1, **R2**(a, c) sur S2. La requête R1 $\bowtie_a$ R2 est posée sur S3. Le schéma du résultat de la requête est (a, b, c). Les attributs sont des entiers positifs. La distribution des valeurs des attributs est uniforme. La taille, en octets, des attributs *a, b, c* est respectivement 1000, 1000 et 3000.
Le tableau suivant donne la taille des résultats intermédiaires en milliard (10^9) d'octets :

Expression	*taille*
R1	80
R2	400
π_a (R1)	10
π_a (R2)	20
R1 $\ltimes_a$ R2	40
R2 $\ltimes_a$ R1	100
R1 $\bowtie_a$ R2	T
π_a (R1 $\bowtie_a$ R2)	5

Remarque : on nomme T la taille du résultat de la requête

Question 1
a) Les affirmations suivantes sont-elles vraies ? Justifier brièvement.

a1) card(R1) = card(π_a (R1))
a2) T < 25 10^9 octets
a3) card(π_a (R1 $\bowtie_a$ R2)) = card(π_a (R1))
a4) card((π_a R1) $\cap$ (π_a R2)) = card(π_a (R2))

b) Donner la valeur numérique des expressions suivantes. Expliquer brièvement.

b1) card(R1)

b2) card(π_a (R1))

b3) le nombre moyen de nuplets de R1 pour une valeur de a donnée.

b4) card(π_a (R1 $\bowtie_a$ R2))

b5) card(R1 $\bowtie_a$ R2)

b6) T

Question 2 :

a) Donner la quantité (en milliard d'octets) de données à transférer pour traiter la jointure selon les plans suivants. Si nécessaire, donner une réponse en fonction de T.

P1) Transférer R1 et R2 vers S3 pour traiter la jointure sur S3

P2) Transférer R1 vers S2 pour traiter la jointure sur S2. Envoyer le résultat vers S3

P3) Transférer R2 vers S1 pour traiter la jointure sur S1. Envoyer le résultat vers S3

P4) Transférer les valeurs de R1.a vers S2, semi jointure sur S2, jointure sur S1, résultat envoyé vers S3.

P5) Transférer les valeurs de R2.a vers S1, semi jointure sur S1, jointure sur S2, résultat envoyé vers S3.

P6) Transférer les valeurs de R2.a vers S1, et celles de R1.a vers S2. Semi-jointures sur S1 et S2, puis jointure finale sur S3.

P7) Sur S1 : transférer les valeurs de R1.a vers S2.
Sur S2 : semi jointure dont le résultat, appelé R3, est transféré vers S3. Transférer également les valeurs de R3.a vers S1.
Sur S1 : semi jointure dont le résultat est transféré vers S3
Sur S3 : jointure finale

P8) Sur S2 : transférer les valeurs de R2.a vers S1.
Sur S1 : semi-jointure dont le résultat, appelé R3, est transféré vers S3. Transférer également les valeurs de R3.a vers S2.
Sur S2 : semi jointure dont le résultat est transféré vers S3
Sur S3 : jointure finale

b) Quel est le plan optimal ?

TD : Transactions réparties

Exercice 1 : Verrouillage réparti de données répliquées

1. Quel est le problème posé lorsqu'on veut verrouiller une donnée répliquée ?
2. Qu'est ce qu'un verrou global , un verrou local ?

.

3. On suppose qu'un gestionnaire de verrou centralise les demandes de verrou et d'accès aux données. Expliquer son fonctionnement en cas de données répliquées. Quels sont les avantages et défauts d'une telle approche ?

4. On cherche maintenant à ne pas mettre en place de gestionnaire centralisé, mais de tirer parti des gestionnaire de verrous et d'accès locaux. Pour cela, les transactions doivent respecter le protocole suivant :

lorsqu'elles ont réussi à verrouiller un certain nombre de répliques d'une donnée, les transactions peuvent déduire qu'elle ont accès à la donnée. On dit qu'elle obtiennent un *verrou logique* sur la donnée grâce aux *verrous physiques* qu'elles ont obtenus sur les répliques. On suppose qu'une donnée *d* est répliquée sur *n* sites.

   a. On suppose que n=3. Montrer qu'en obtenant au moins 2 verrous physiques exclusifs, une transaction peut avoir un verrou logique exclusif sur *d* et donc accès en écriture à *d*, donc à toutes ses répliques.
   b. Soit k (resp. k') le nombre de verrous physiques exclusifs (resp. partagés) à obtenir pour en déduire un verrou logique exclusif (resp. partagé). Donner la valeur minimale de k (resp. k') en fonction de n que le protocole doit respecter pour que le verrouillage logique fonctionne.

Exercice 2 : sérialisation globale et locale

On suppose les objets x et y stockés sur le site S1, et les objets z et w sur le site S2. Déterminer pour chacune des exécutions suivantes de deux transactions Ti et Tj les graphes de précédence locaux et global et dire si elle est sérialisable ou non.

1) Exécution 1 :
   S1 : Li(x), Lj(x), Ej(x), Ei(x)
   S2 : Li(w), Lj(z), Ej(w), Ei(w)

2) Exécution 2 :
   S1 : Li(x), Ei(x), Lj(x), Ej(x), Li(y), Ej(y)
   S2 : Lj(z), Ej(z), Li(z), Ei(w)

3) Exécution 3 :
   S1 : Li(y), Lj(x), Ej(x)
   S2 : Ei(z), Li(w), Lj(w), Ej(w)

Exercice 3 : détection d'interblocage

A et B sont stockés sur S1, C et D sur S2. On utilise un verrouillage en deux phases. On exécute les transactions T1 à T9. Les demandes d'accès sur les différents granules sont les suivants :

S1 A : L7, E3, E2, L1
B: L3, L4, E6, L9

S2 C: L9, E8
D : E8, L7, L5

1) Quelles sont les transactions globales ?
2) Décrire l'évolution de la table des verrous dans le cas où une demande de verrou partagé sera toujours satisfaite si possible. Représenter la table avant la première libération de verrou possible. Rappeler quel est l'inconvénient d'une telle gestion.
3) Proposer une autre politique permettant de résoudre le problème précédent et montrer l'état des tables de verrous dans ce cas.
4) Construire les graphe d'attente locaux et le graphe d'attente global. Est-il nécessaire d'utiliser les premiers en entier pour construire le dernier ? Que concluez-vous du résultat obtenu ?
5) On centralise la détection d'interblocages sur S1. Comparer deux solutions : dans la première, chaque modification du graphe local d'attente de S2 est communiqué à S1. Dans la seconde, la copie complète du graphe local de S2 est communiqué périodiquement à S1.

Les Index

TME : Les Index

L'objectif de ce TP est de mettre en évidence l'utilisation des index pour améliorer le temps de réponse des requêtes. Lire l'annexe **PlanRequete**

Installer les fichiers du TME

Ouvrir une fenêtre de terminal (xterm), pour exécuter les commandes suivantes :

commande	description
cd	aller dans votre répertoire $HOME
tar zxvf $BD_TOOL/tp-index.tgz	installer l'archive dans votre répertoire principal
cd tp-index	aller dans votre répertoire de travail
emacs annuaire.sql &	éditer le fichier
SQL> @annuaire	exécute le script annuaire.sql depuis l'invite SQL

Exercice

1) (préparation) Créer une relation Annuaire(nom, prenom, age, cp, tel) en utilisant le fichier *annuaire.sql*.

```
SQL> @annuaire
```

Elle contient 10000 tuples sur 90 prénoms et 100 noms différents. Les codes postaux (cp) sont des multiples de 100 et sont compris entre 1000 et 100000. Le numéro de téléphone est une chaîne de 10 chiffres commencant par 0. Le numéro de téléphone est une clé de l'annuaire.

Répondre aux questions suivantes dans le fichier tme2.sql

2) Proposer une ou plusieurs requêtes pour vérifier si la distribution de l'attribut *age* est uniforme, quasi uniforme ou fortement biaisée. Expliquer brièvement comment analyser le résultat de cette requête.

3) Proposer une ou plusieurs requêtes pour vérifier que les attributs *age* et *prénom* sont indépendants :

- Quel que soit l'age, y a-t-il autant de prénom différents pour un age donné ?
- Quel que soit le prénom, y-a-t-il autant d'âge différents pour un prénom donné ?
- Y-a-t-il tous les prénoms possibles pour chaque age, et tous les ages possibles pour chaque prénom ?

4) En utilisant les formules du cours, estimer la cardinalité des requêtes suivantes. Comparer ensuite la cardinalité estimée avec la cardinalité réelle des requêtes. Quel est le pourcentage d'erreur (réel/théorique) ?

- R1: select * from Annuaire

- R2: select * from Annuaire where age < 65;
- R3: select * from Annuaire where cp between 75000 and 78000;
- R4: select * from Annuaire where age=18 and cp < 1200;
- R5: select * from Annuaire where age=18 and cp = 1200;
- R6: select count(*) from Annuaire where age < 65;

5) Créer les index mono-attributs permettant d'améliorer les performances des requêtes précedentes. La syntaxe est :

- **create index** *nom_index* **on** *nom_relation*(*nom_attribut*) ;
- pour effacer un index : **drop index** *nom_index*;

5.1) Afin de visualiser les informations détaillées sur le traitement des requêtes :

- activer le mode de visualisation des plans :
  - set autotrace trace explain stat (pour désactiver ce mode : set autotrace **off**)
- Dans cette question, on demande à l'optimiseur de requêtes d'utiliser des règles heuristiques et non pas une estimation du coût. La session est modifiées en conséquence :
  - alter session set optimizer_mode = **RULE**;
- puis re-exécuter chaque requête pour voir le plan d'exécution. Voir l'annexe **PlanRequete**.

Pour chaque requête, afficher les index utilisés par le moteur de requête.

Présenter vos réponses dans un tableau récapitulatif :

requête	**index utilisé(s)**
R1	...
...	...

Analyser le tableau et suggérer des règles heuristiques que l'optimiseur utilise pour choisir un index. Si nécessaire, compléter le tableau avec d'autres requêtes.

5.2) Quelles sont les requêtes pour lesquelles le plan d'exécution construit par l'optimiseur ne vous semble pas optimal ? Donner un exemple.

6) Estimation du coût des accès. L'ordre ANALYZE ajoute des statistique dans le dictionnaire du SGBD pour permettre à l'optimiseur d'estimer le coût des requêtes :

- analyze table Annuaire compute statistics; (pour supprimer les statistiques: analyze table Annuaire delete statistics;)

6.1) En interrogant les vues user_tables, user_tab_columns et user_indexes, expliquer quelles sont les statistiques maintenues dans le dictionnaire. Désactiver le mode autotrace avant d'interroger les vues (set autrotrace off).

- parmi les attributs de ces vues, quels sont ceux qui représentent les statistiques présentées en cours ?
  - cardinalité d'une relation
  - description du domaine d'un attribut avec ses valeurs min et max et son nombre de valeurs distinctes,
  - etc...

6.2) Afin que le SGBD maintienne des statistiques sur vos index, exécutez la commande suivante pour chaque index :

- analyze index *nom_index* compute statistics;
- On demande maintenant à l'optimiseur de requêtes d'utiliser un modèle de coût (et non pas des règles heuristiques) pour choisir un plan d'exécution efficace. La session est modifiées en conséquence :
  - alter session set optimizer_mode = **CHOOSE**;

Pour chaque requête, afficher les index utilisés par le moteur de requêtes. Expliquer pourquoi les index utilisés ne sont plus les mêmes qu'en 5.1)

6.2.1) En déduire la méthode que l'optimiseur utilise pour choisir un index.

6.2.2) Est ce l'optimiseur estime avec précision la cardinalité des requêtes ? Si non, expliquer pourquoi.

6.3) On veut déterminer à partir de quel facteur de sélectivité, l'optimiseur choisit la lecture séquentielle plutôt que l'accès par index. On appelle *S* le seuil du facteur de sélectivité au delà duquel la lecture séquentielle est utilisée de préférence à l'index. Combien vaut S ?

- Méthode suggérée pour déterminer S :
  - Faire varier la sélectivité du prédicat $cp < v$ en faisant varier v, puis déterminer par dichotomie la valeur v à partir de laquelle l'index n'est plus utilisé.
- Mesurer S pour une requête contenant 2 prédicats de sélection. S est-il toujours le même? pourquoi ?

6.4) Quelles sont les requêtes pour lesquelles le plan d'exécution construit par l'optimiseur n'est pas optimal, donner un exemple.

7) (facultatif) Proposer une méthode pour mesurer la dégradation de performance due à l'index lors de l'insertion de données dans l'annuaire.

---

retour vers: **LesTravauxDirigés**, **LesCours**, l'**Accueil**.

Dernière modification le mars 6, 2009 12:02 .

Éditer | Déverrouiller la Page | Supprimer la Page | Historique | Diff | InfosDeLaPage | DebugInfo

Vous êtes connecté en tant que **bdr** | Déconnexion

Les Jointures

TME Optimisation des requêtes avec jointures

L'objectif de ce TME est de trouver le plan d'exécution optimal pour traiter une requête contenant des jointures. Ce TME met en pratique les notions du cours sur l'optimisation de requêtes: plan d'exécution, espace de recherche, ordonnancement des jointures, modèle de coût, algorithme de jointure.

Sujet

- voir l'exercice 3 du poly de TD (base Joueur, Club, Finance)

Documentation

- Signification des valeurs mesurées pendant l'exécution d'une requête : **Annexe1**
- Directives d'optimisation: **Annexe2**
- Documentation annexe: L'optimiseur de requêtes d'Oracle.

Installation des fichiers du TME

- tar zxvf $BD_TOOL/tp-jointure.tgz
- cd tp-jointure
- emacs **tme3.sql** &

Préparation

- construire la BD de l'exercice 3
  - @base3
- ajouter les index
  - @index3
- activer la visualisation des plans d'exécution
  - set autotrace trace explain stat

Introduction

Dans la suite du TME, nous mesurons le **coût d'une requête** en nombre de lectures de blocs mémoire. C'est le nombre de **consistent gets** qui apparait dans les statistiques d'exécution d'une requête.

Un **plan d'exécution** est affiché de manière arborescente. Chaque noeud de l'arbre est un opérateur. Un opérateur a un numéro de noeud et un nom. Les noms des opérateurs sont **indentés** pour montrer l'arbre d'exécution : le noeud parent est celui dont le nom est décalé d'**un** caractère à gauche.

- l'exemple suivant représente une jointure par boucles imbriquées (nested loops). L'opérateur de jointure (noeud numéro 1) est le fils du noeud numéro O. Les fils du noeud numéro 1 sont

des lectures séquentielles.

```
0  SELECT
1   NESTED LOOPS
2    TABLE ACCESS
3    TABLE ACCESS
```

- Autre exemple : l' accès à la relation **C** (les clubs) par un index sur l'attribut **division** :

```
3  TABLE ACCESS (BY INDEX ROWID) OF 'C'
4   INDEX (RANGE SCAN) OF 'I_C_DIVISION' (NON-UNIQUE)
```

Les opérateurs implémentés par le moteur de requêtes sont :

nom	algorithme
nested loops	jointure par boucles imbriquées
merge join	jointure par fusion
hash join	jointure avec hachage temporaire des tuples
sort (join)	tri préliminaire avant jointure par fusion
table access full	lecture séquentielle d'une relation
table access by rowid	lecture non séquentielle d'une relation (un accès par tuple)
index (range scan)	traversée d'un index (arbre B+)

IMPORTANT

Dans la suite de ce TME on demande au SGBD ne réaliser des jointures par boucles imbriquées (Nested Loop Join : voir diapo 13 du cours).
Ajouter la directive **USE_NL**(x,y,z,...) dans toutes les requêtes (x,y,z sont les tables qu'on veut joindre par boucles imbriquées, ici x=C, y=J, z=F)

Question 1 : Plan d'exécution choisi par l'optimiseur

1.1) Représenter l'arbre du plan d'exécution **P** choisi par défaut par l'optimiseur pour traiter la requête R?

1.2) Mesurer le coût du plan P (cf. **Annexe1**).

Question 2: Ordonancement des jointures

L'objectif est de trouver l'ordre optimal pour traiter les jointures.

2.1) Quelles relations de la BD peuvent être jointes avec J par equi-jointure ? Même question pour C et F. Combien y a-t-il d'ordres de jointure différents pour traiter la requête R ?

2.2) En utilisant la directive /*+ use_nl(c,j,f) **ordered** */, construire un plan d'exécution pour chaque ordre de jointure. L'ordre est celui de la clause FROM (pas celui de la clause where !). Pour chaque plan, préciser clairement les index et les algorithmes de jointure utilisés, et mesurer le coût du plan. Lire l'**Annexe2** expliquant la directive ordered.

2.3) Quels sont les 3 plans de plus faible coût ? Comparer leur coût avec celui du plan **P** choisi par l'optimiseur ?

Question 3 : Coût des sélections

On veut connaitre le coût des sélections utilisées dans la requête R. Pour cela, on mesure le coût des sélections avec et sans index.

3.1) Quels sont les noms des index existants pour les relations J, C et F ?

3.2) Pour chaque requête de sélection pouvant servir à traiter la requête R, donner son expression SQL et son coût avec et sans index. Pour empêcher un accès par index, utiliser la directive /*+ no_index(*nom_relation nom_index*) */ (voir l'**Annexe2**). Donner aussi la cardinalité (nb de n-uplets) du résultat de la sélection.

num	sélection	SQL	coût avec index	coût sans index	cardinal.
S1	sel(J, salaire > 59000 et sport = 'sport1')				
S2	sel(C, division=1)				

3.3) Pour les sélections S1, S2 l'accès par index est-il avantageux ? Expliquer pourquoi.

3.4) On veut traiter R en commençant par la sélection la moins couteuse (entre S1 et S2). Donner le plan de la requête R, qui utilise la sélection la moins couteuse? Est-ce le plan optimal ?

Question 4

(question supprimée)

Question 5 : Optimisation basée sur un modèle de coût

5.1) Ajouter dans le dictionnaire du SGBD les statistiques sur J, C et F et les index associés.

- lire et comprendre le fichier stat3.sql, puis l'exécuter:
- @stat3

5.2) Lorsque le dictionnaire du SGBD contient des statistiques sur les données lues par la requête, l'optimiseur d'Oracle peut estimer plus précisément le coût des opérations d'un plan d'exécution. Cela lui permet de déterminer si un accès par index est intéressant ou non. Modifier le prédicat de sélection du salaire dans la requête R (incrémenter de 100 en 100, la valeur 59000) jusqu'à ce que le SGBD utilise l'index. On note **R'** la requête obtenue.

5.3) Supprimer les statistiques, la commande est :

- analyze table J **delete** statistics;
- *(idem pour les 2 autres tables)*

Le plan choisi par l'optimiseur pour traiter **R** est-il le même **avec** et **sans** statistiques sur J,C,F ? Expliquer pourquoi.
Décrémenter de 1000 en 1000, la valeur 59000, jusqu'à ce que le SGBD n'utilise plus l'index. On note **R''** la requête obtenue.

5.4) Lorsque l'optimiseur ne dispose pas de statistiques, il tente tente d'estimer la taille des résultats intermédiaires à partir d'un échantillon de données (technique appellée *dynamic sampling*). Désactiver le dynamic sampling en ajoutant la directive suivante.

```
select /*+ use_nl(c,j,f) dynamic_sampling(0) */ c.nom, f.budget
```

a) Expliquer pourquoi le plan généré par l'optimiseur est différent.
b) Dessiner l'arbre du plan d'exécution. Comprendre les opérations effectuées. L'optimiseur choisit de calculer une intersection entre deux ensembles. Quels sont ces 2 ensembles ?

Question 6 (facultatif)

6.1) Laisser le SGBD choisir les algorithmes de jointure, en supprimant la directive use_nl. Pourquoi la jointure par hachage est-elle choisie ?

6.2) Insérer des nuplets dans la table C jusqu'à ce que l'algorithme de jointure choisi change (pour un autre algorithme que le hash join)

Question 7 (facultatif)

7.1) Ajouter une 4eme table et proposer une requête de jointure entre les 4 tables qui ne puisse pas être transformée en un arbre linéaire gauche.

7.2) Ajouter suffisamment de données dans les tables pour qu'au moins une jointure par tri-fusion (sur les 3 jointures à effectuer) soit choisie par l'optimiseur.

Question 8 (facultatif)

Donner un exemple de requête R' de jointure entre C,J,F où le plan choisi par le SGBD consiste à commencer par une lecture séquentielle de F en entier. Cependant, ce plan serait efficace car la sélectivité de première jointure entre F et une autre relation serait très forte. R' est différent de R, vous pouvez changer les prédicats de la clause *where*.

---

retour **LesTravauxDirigés**, **Accueil**,

Dernière modification le mars 17, 2009 6:09 .
Éditer | Déverrouiller la Page | Supprimer la Page | Historique | Diff | InfosDeLaPage | DebugInfo
Vous êtes connecté en tant que **bdr** | Déconnexion

Annexe 1

Annexe1: Valeurs mesurées pendant l'exécution d'une requête

Parmi les valeurs mesurées, nous considérons **uniquement** le nombre de **consistent gets**, sans prendre en compte les autres valeurs, car **consistent gets** est représentatif du nombre de pages disque lues (dans un état *froid* de la base, lorsque toute les données sont sur disque et que le cache est vide).

Les principales valeurs mesurées sont:

Statistics
.----------

- db block gets : Number of times a CURRENT block was requested
- **consistent gets** : Number of times a consistent read was requested for a block.
- physical reads : Total number of data blocks read from disk. This number equals the value of "physical reads direct" plus all reads into buffer cache.
- redo size : Total amount of redo generated in bytes.
- sorts (disk) : Number of sort operations that required at least one disk write. Sorts that require I/O to disk are quite resource intensive. Try increasing the size of the initialization parameter SORT_AREA_SIZE. For more information, see "SORT_AREA_SIZE".
- sorts (memory) : Number of sort operations that were performed completely in memory and did not require any disk writes. You cannot do much better than memory sorts, except maybe no sorts at all. Sorting is usually caused by selection criteria specifications within table join SQL operations.
- rows processed: result cardinality

---

retour au TME **LesJointures**

Annexe 2

Annexe2: Directives d'optimisation

Les directives d'optimisation guident l'optimiseur de requêtes pour le choix d'un plan d'exécution.

Syntaxe: les directives sont insérées **dans** la requêtes, après le mot SELECT, entre commentaires spéciaux :

/*+ *nom_directive* */

Utiliser les directives d'optimisation (hint) suivantes :

- /*+ ordered */ : fixe l'ordre de traitement des jointures : l'ordre imposé est celui indiqué dans la clause **FROM** de la requête.
- /*+ index(*nom_relation nom_index*) */ : pour forcer un accès par index
- /*+ no_index(*nom_relation nom_index*)*/ : pour interdire l'utilisation d'un index

Ne pas confondre le **nom de l'index** et le nom de l'attribut indexé. Par exemple, I_J_cnum est le nom de l'index sur l'attribut cnum de J.

Exemple de directive pour ordonner les jointures

Le plan P1 avec l'ordre (J |><| C) |><| F

```
select /*+ ordered */ c.nom, f.budget
from J, C, F
where J.cnum = C.cnum and C.cnum = F.cnum
and c.division=1 and J.salaire > 59000
and j.sport = 'sport1';
```

---

retour vers **LesJointures**

Dernière modification le février 26, 2004 9:14 .
Éditer | Déverrouiller la Page | Supprimer la Page | Historique | Diff | InfosDeLaPage | DebugInfo
Vous êtes connecté en tant que bdr | Déconnexion

Jointure Répartie

TME Jointure Répartie

L'objectif de ce TME est de comprendre l'évaluation d'un requête de jointure entre 2 relations qui sont situées sur 2 sites distincts.

- définir le schéma global qui offre un accès transparent à des données de plusieurs bases,
- formuler une requête répartie,
- comprendre l'ordre et l'emplacement des opérations permettant d'évaluer une requête répartie (quel site traite quelles opérations?).

Scénario

On dispose de 2 SGBD : site1 et site 2
Données: Le site 1 contient les clubs (table C), le site 2 contient les Joueurs (table J)
La couche BDR est implémentée sur le site 1.

Installation

- créer les tables J,C,F (déjà fait lors du TME précédent)

```
@base3
```

- supprimer les Joueurs du site 1 (les joueurs sont maintenant sur le site 2)

```
drop table J;
```

- Relier les sites :
  - La couche BDR (site1) doit pouvoir se connecter au site 2

```
create database link ora2 connect to bdwa2 identified by "bdwa2" using 'ora2';
```

- Vérifier le bon fonctionnement du lien

```
desc J@ora2
```

ajouter un club dans une nouvelle ville. Ce club n'a que 10 joueurs ce qui permettra, par la suite, de poser une requête de jointure très sélective.

```
insert into C values( 6000, 2, 'petit club', 'Combourg');
```

Construire le schéma global

```
create view J as
select * from j@ora2;
```

Requêtes réparties

Pour chaque requête, répondre aux questions

- Où est traitée chaque opération (sélection, projection, jointure, ...) ?
- Quelles sont les données transférées entre les sites pendant l'évaluation de la requête ?

- Activer le mode de visualisation des plans et le chronométrage

```
set timing on
set autotrace trace explain stat
```

R1 : Jointure seule avec un transfert volumineux

- Afficher les joueurs avec leur club

```
select *
from J, C
where j.cnum = c.cnum;
```

R2 : jointure avec sélection

```
select *
from J, C
where j.cnum = c.cnum
and salaire > 59000
```

La sélection est-elle poussée sur le site 2 ?

R3 Jointure très sélective

- R3a : jointure très sélective et avec un transfert volumineux

```
select *
from J, C
where j.cnum = c.cnum
and ville = 'Combourg';
```

- R3b : jointure très sélective et avec un transfert faible

```
select /*+ driving_site(j1) */ *
from J j1, c c1
where j1.cnum = c1.cnum
and ville = 'Combourg';
```

- Proposer d'autres requête pour illustrer les optimisations de requêtes réparties vues en cours.

LesTravauxDirigés, **Accueil**

Dernière modification le mars 13, 2009 12:23 .

TME - Architecture Web

JDBC

Introduction:

L'objectif de ce TD/TME est de réaliser une application de type web qui accède à une base de données. Cette application repose sur l'architecture client-serveur à 3 couches (*cf.* cours sur l'architecture C/S 3-tiers) : le serveur de données, le serveur web et le navigateur client. Nous construisons la **passerelle d'accès au serveur de données avec JDBC.**

Le serveur de données (Oracle sur la machine oracle), est un SGDB relationnel supportant l'interface JDBC. Il contient la base de données tennis dont le schéma est:

JOUEUR2 (NUJOUEUR, NOM, PRENOM, ANNAISS, NATIONALITE)

GAIN2 (NUJOUEUR, LIEUTOURNOI, ANNEE, PRIME, SPONSOR**)**

RENCONTRE2 (NUGAGNANT, NUPERDANT, LIEUTOURNOI, ANNEE, SCORE**)**

Les attributs NuGagnant, NuPerdant et NuJoueur sont définis sur le même domaine. Les clés des relations sont soulignées.

Etape 1 : Accès à une base de données avec JDBC

Le package JDBC permet d'accéder au SGBD depuis une application écrite en langage java. Afficher la documentation en ligne du TME (lien TME JDBC depuis la page d'accueil), puis afficher la documentation java (lien bibliothèque JDBC).

Utilisation des bibliothèques Java

Les bibliothèques java sont regroupées en packages. Un package contient plusieurs interfaces et plusieurs classes. Une classe (et une interface) contient des méthodes et des variables. La documentation en ligne permet de naviguer dans les bibliothèques pour déterminer les classes, interfaces et méthodes à utiliser pour construire une application java. Les classes et interfaces JDBC du package ***java.sql*** permettent d'accéder à une base de données. Naviguer dans la documentation du package java.sql pour répondre aux questions suivantes :

a) Donner le nom et le type des paramètres des méthodes getConnection de la classe DriverManager.

b) A quoi sert la méthode *next* de l'interface ResultSet ?

c) A quoi sert la méthode *setMaxRows* de l'interface Statement ?

d) Quelles sont les sous-interfaces de l'interface Statement ?

1 Première connexion à une base de données

Installer l'environnement logiciel (la procédure d'installation est détaillée sur la fiche technique, ci après) puis exécuter le programme Joueur.
1.1) Etudier le programme Joueur.java. (voir en annexe). Commenter brièvement les lignes importantes pour expliquer chaque étape du programme.
1.2) Sur une feuille (à faire chez soi), représenter sous la forme d'un graphe, les scénarios d'accès à une base de donnée en utilisant les interfaces, classes et les méthodes de JDBC. Le graphe est défini comme suit :

- un **nœud** (rectangle) représente une interface ou une classe.
- un **arc** orienté (flèche) représente une méthode. L'arc est tel que :
- son origine est l'interface dans laquelle la méthode est définie,
- sa destination est l'interface du résultat de la méthode.

Le graphe doit contenir les classes, interfaces et méthodes du package java.sql qui sont utilisées dans *Joueur.java*, ainsi que les éléments suivants : PreparedStatement, ResultSetMedatata, DatabaseMetadata, executeUpdate, int, String et getMetaData.

2 Accès paramétré à la base de données

2.1) Exécuter le programme MaxPrime (saisir une année, par ex. 1992). Editer le fichier MaxPrime.java. Compléter l'en-tête avec vos nom et prénom. Compléter tous les commentaires pour expliquer ce que fait le programme.
2.2) Copier et modifier le programme précédent pour obtenir le programme *MaxPrime2.java* qui exécute la même requête **en boucle,** en demandant à chaque itération une nouvelle valeur à l'utilisateur. Le programme MaxPrime2 se termine lorsque la saisie de l'utilisateur est vide. Pour améliorer les performances du programme, deux conditions sont requises :

- la connexion vers le SGBD est ouverte une seule fois pendant toute la durée du programme (*i.e.,* réutiliser le même objet de type Connection à chaque itération).

- le SGBD analyse la requête une seule fois pendant toute la durée du programme (*i.e.,* utiliser l'interface PreparedStatement pour préparer une requête paramétrée ; utiliser la méthode setInt pour affecter une valeur à un paramètre de la requête).

Remarque : ne pas oublier d'appeler le constructeur de MaxPrime2. (*ie*., remplacer new MaxPrime() par new MaxPrime2().

3 Requête générique

3.1) Quel est le rôle de l'interface ResultSetMetaData et de sa méthode getColumnCount ?

3.2) Compléter le programme *Generique.java* pour traiter une requête SQL quelconque passée en paramètre, et afficher les valeurs des tuples du résultat sous la forme « val_1 val_2 … val_n » (un tuple par ligne).

Tester l'exécution avec la requête donnant tous les Joueurs nés en 1972 :

```
java Generique "select * from Joueur2 where annaiss = 1972"
```

3.3) Compléter le programme pour afficher en en-tête le **nom des attributs** du résultat.

Exemple d'exécution (ne pas chercher à tabuler le résultat)

```
NUJOUEUR    NOM         PRENOM      ANNAISS     NATIONALITE
-----------------------------------------------------------
1           MARTINEZ    Conchita    1972        Espagne
14          SAMPRAS     Pete        1972        Etats-Unis
```

3.4) Ecrire le programme *GeneriqueXHTML.java* qui affiche le résultat d'une requête quelconque dans un tableau formaté en XHTML. Tester l'exécution en stockant le résultat dans un fichier :

```
java GeneriqueXHTML "select * from Joueur2" > resultat.html
```

Exemple de résultat :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html>
  <head ><title>Résultat</title></head>
<body>
  <h3>La requete est : </h3> select * from Joueur2
  <h3>le resultat est : </h3>
  <table border="2">
   <tr><th>NUJOUEUR</th><th>NOM</th><th>PRENOM</th><th>ANNAISS</th><th>NATIONALITE</th></tr>
   <tr><td>1</td><td>MARTINEZ</td><td>Conchita</td><td>1972</td><td>Espagne</td></tr>
   <tr><td>2</td><td>NAVRATILOVA</td><td>Martina</td><td>1957</td><td>Etats-Unis</td></tr>
   …
   <tr><td>14</td><td>SAMPRAS</td><td>Pete</td><td>1972</td><td>Etats-Unis</td></tr>
  </table>
</body>
</html>
```

Visualiser graphiquement le résultat dans un navigateur avec la commande :

```
netscape resultat.html (ou mozilla resultat.html)
```

4 Schéma d'une relation

4.1) Quel est le rôle de l'interface DatabaseMetaData et comment obtenir une instance de ce type à partir d'une connexion ? Expliquer la méthode getColumns de l'interface DatabaseMetaData. Dans un SGBD les relations sont généralement organisées en hiérarchie à 2 niveaux (catalogue et schéma). Ainsi, une relation appartient à un schéma lui-même appartenant à un catalogue. Dans Oracle, l'organisation a un seul niveau : une relation appartient à un schéma égal au nom de celui qui a créé la relation, il n'y a pas de niveau catalogue.

4.2) Créer le programme Schema.java qui affiche le nom et le type des attributs d'une relation passée en paramètre.

Exemple d'exécution

```
java Schema JOUEUR2                    // écrire le nom de la relation en MAJUSCULE
Le schéma de JOUEUR2 est :             // résultat affiché
NOM          TYPE
------------------------------
NUJOUEUR     NUMBER
NOM          VARCHAR2
PRENOM       VARCHAR2
ANNAISS      NUMBER
NATIONALITE        VARCHAR2
```

5 Jointure inter-bases

Soit une deuxième base de données (située sur un autre SGBD différent du premier), contenant la relation

SPONSOR (NOM, NATIONALITE) // nationalité des sponsors

La deuxième base de données est accessible seulement en lecture, par une connexion à la base oracle en tant qu'utilisateur «anonyme» avec le mot de passe «anonyme».La relation Sponsor contient **100 000** tuples. Soit la requête R1 :«Donner le nom et la nationalité des joueurs avec le nom et la nationalité de leurs sponsors, dans l'ordre des noms de joueur».

5.1) Quelle est la durée approximative d'une lecture séquentielle de la relation Sponsor ? Répondre en écrivant le programme *Generique2.java* (obtenu en modifiant *Generique.java*), puis en utilisant la commande *time* :

```
time java Generique2 "requete"...      // mesure le temps de réponse de la requête
```

Veiller à omettre l'affichage du résultat dans le terminal pour éviter de mesurer le temps d'affichage au lieu du temps de lecture des données.

5.2) Ecrire R1 en SQL.

5.3) Pourquoi ne peut-on pas exécuter cette requête R1 avec une seule connexion au SGBD ?

5.4) On veut obtenir la cardinalité du résultat de R1 ? Donner en SQL une requête R2 telle :

R2 est une sous-expression de R1, et R2 peut être exécutée entièrement sur le premier SGBD

la cardinalité de R2 est égale à celle de R1, *i.e.,* card(R2) = card(R1)

Exécuter R2 et donner la valeur de card(R1).

5.5) Compléter le programme Sponsor.java pour traiter R1. Pendant tout le traitement de la requête R1, combien d'instances de type Connection et Statement sont créées ? Combien de sous-requêtes sont exécutées ? Combien de fois la relation Sponsor est-elle lue ? Quels sont les tuples transférés depuis le SGBD vers l'application java ?

Exemple d'exécution :

```
java Sponsor                           //commande
CONNORS, Etats-Unis, Dunlop, Ecosse
CONNORS, Etats-Unis, Lacoste, France
EDBERG, Suede, Dunlop, Ecosse
…
SAMPRAS, Etats-Unis, Reebok, Angleterre
WILANDER, Suede, Dunlop, Ecosse
WILANDER, Suede, Kennex, USA
```

5.6) Mesurer le temps moyen d'exécution du programme avec la commande time :

```
time java Sponsor                              //commande
…
real 0m3.957s          user 0m1.530s           // affiche le temps de réponse
```

5.7) Ecrire le programme SponsorTF.java pour traiter la requête R2 : «Donner le nom et la nationalité des joueurs avec le nom et la nationalité de leurs sponsors, dans l'ordre des **noms de sponsor**» en utilisant l'algorithme de jointure par tri fusion. Combien d'instances de type Connection et Statement sont créées pendant le traitement de la requête ?

5.8) Comparer les temps de réponse de R1 et R2. Mesurer (en pourcentage) l'écart entre le temps de réponse de Sponsor et SponsorTF. Interpréter le résultat. Proposer une solution Sponsor2.java pour améliorer le temps de réponse de la requête R2.

5.9) Détailler une solution pour traiter R1 en transférant les clés (voir cours : semi-jointure)

5.10) (facultatif) Proposer une implémentation pour d'autres algorithmes de jointure (grace join, hybrid hash join, ...).

Fiche Technique pour les TME

Installer l'environnement logiciel (Java, JDBC)

1.1) Répertoire de travail.

Pour faciliter l'utilisation des divers outils, le nom des répertoires et de vos fichiers de travail doivent être identiques pour tous les binômes. Pour créer l'arborescence de travail, sauvegarder l'archive *jdbc.tar* dans votre **répertoire racine** et extraire son contenu.

```
cd                                  // aller dans le répertoire $HOME
tar zxvf $BD_TOOL/jdbc-etu.tgz      // installer l'archive
cd jdbc                             // aller dans le répertoire de travail
```

Dans la suite du TME, sauvegarder tous vos programmes dans le répertoire ~/jdbc.

1.2) Editeur de texte:

Editer les fichiers avec emacs. Activer les options suivantes :

Programme en couleur : Menu Option > Syntax Highlighting

Repérage de la structure du programme : Menu Option > Paren Match Highlighting

1.3) Environnement java

Pour tester la compilation du code source en bytecode, compiler le programme de test Bonjour.java (cela crée le fichier Bonjour.class)

```
javac Bonjour.java
```

Pour tester la machine virtuelle lancer l'exécution du fichier Bonjour.class :

```
java Bonjour
```

Fichier Joueur.java

```
1  import java.sql.*;
2  import java.io.*;
3
4    public class Joueur {
5      String server = "frelon";
6      String port = "1521";
7      String database = "oracle";
8      String user = "p6lip000";
9      String password = "p6lip000";
10     String requete = "select nom, prenom from Joueur";
11     Connection connexion = null;
12     …
13     /** La méthode traiteRequete */
14     public void traiteRequete() {
15         try {
16            String url = "jdbc:oracle:thin:@" +server + ":" +port + ":" +database;
17            connexion = DriverManager.getConnection(url, user, password);
18            Statement lecture =  connexion.createStatement();
19            ResultSet resultat = lecture.executeQuery(requete);
20            while (resultat.next()) {
21              String tuple = resultat.getString(1) + " " + resultat.getString(2);
22              out.println(tuple);
23            }
24            resultat.close();
25            lecture.close();
26            connexion.close();
27         }
28         catch(Exception e){ … }
29     } }
```

JDBC

TME JDBC : Accès à un SGBD depuis Java

Préparation

- lire le sujet (ou voir poly). Seulement la partie JDBC (pas JSP)
- lire les API (choisir le package java.sql)
- installer les fichiers : tar zxvf $BD_TOOL/jdbc-etu.tgz
- ceux qui utilisent Eclipse doivent référencer le jar /Infos/bd/client10/ojdbc14.jar dans leur projet.

Séance 1

- installer l'environnement de travail (voir le poly, étape 1 uniquement)
- Commencer l'étape 1 du sujet: répondre aux questions a) à d) dans le fichier rapport.txt
- question 1.1 : compléter les lignes /* commentaires: */

Question fréquente concernant la compilation java

- Comment éviter les warning du compilateur concernant le jeu de caractères UTF-8 ?
  - afin de pouvoir compiler des programmes java contenant des caractères accentués, le jeu de caractères par défaut doit être européen (ISO-8859-1) :
  - export LC_CTYPE="fr_FR" *ou* export LC_ALL="fr_FR"
  - vérification du jeu avec la commande : locale charmap

Question 2 : MaxPrime2. Définir une requête *paramétrée* :

- la requête est une chaîne de caractères contenant un point d'interrogation **?** pour chaque paramètre
- Exemple "select * from Joueur2 where annaiss = **?** "
- voir l'exemple dans la documentation de l'interface PreparedStatement

Questions fréquentes concernant MaxPrime2:

- Comment **comparer 2 chaînes** de caractères a et b ?
  - utiliser la méthode : a.equals(b)
  - l'opérateur a==b ne compare pas le contenu des chaînes de caractères mais leur identifiant d'objet.
- Comment **convertir** une chaîne de caractères en un nombre entier ?
  - int n = Integer.parseInt(chaine);

Questions 3 : Requête générique

- L'exécution du programme GeneriqueHTML doit produire ce résultat

Séance 2

- Finir la question 3
- Question 4: Schéma d'une relation.
- Question 5:
  - Ajuster l'ULR d'accès à la 2eme base
    - String url2 = "jdbc:oracle:thin:@oracle2.ufr-info-p6.jussieu.fr:1521:ora2"; // base des sponsors
  - jointure inter-bases
  - jointure par boucles imbriquées

Séance 3

- Finir la question 5
  - jointure par tri puis fusion. Etendre l'algorithme pour traiter le cas d'une équi-jointure entre 2 attributs non uniques (*i.e.* 2 clés étrangères).
  - jointure par transfert de clés (semi-jointure).
  - (facultatif) implémenter d'autres algorithmes de jointure, tout en utilisant JDBC.

Documentation diverse

- Algorithmes de jointure (anglais)
- Liens externes : un cours HTML (université de Nice), un autre manuel HTML, un cours java, Java 1.5 API, ...

---

retour vers: **TmeJSP**, **LesTravauxDirigés**, **LesCours**, l'**Accueil**.

Dernière modification le avril 7, 2009 5:01 .

Éditer | Déverrouiller la Page | Supprimer la Page | Historique | Diff | InfosDeLaPage | DebugInfo

Vous êtes connecté en tant que **bdr** | Déconnexion

Tme 2 PC

Transactions réparties : validation en 2 étapes (2 Phases Commit)

Préparation

- Voir le cours: protocole 2PC
- Installer les fichiers du tme:
  - tar zxvf $BD_TOOL/tme-2PC.tgz
  - cd tme-2PC
  - source config-simjava
  - javac Exemple1.java
  - appletviewer exemple1.html

Dans les exercices suivants recopier les fichiers java et html :

- cp Exemple1.java Exemple*NN*.java
- cp exemple1.html exemple*NN*.html

puis éditer les nouveaux fichiers pour **remplacer** le nom de la classe Exemple1 par Exemple*NN*

Veiller à **recompiler** votre exemple avant chaque exécution car les noms de classes *Participant* et *Coordinateur* existent dans plusieurs exercices.

Documentation

- la doc java et simjava (voir les classes Sim_entity et Sim_event du package eduni.simjava)

Exercice 1 : Prise en main du simulateur

- Dérouler la simulation à faible vitesse (*speed entre 100 et 150*)
- Expliquer brièvement le fonctionnement du protocole implémenté dans le fichier Exemple1.java
- Comprendre les méthodes utilitaires de la classe Entité : envoyer_message, attendre_message_timeout, panne, date, ...

Exercice 2 : Tolérance aux pannes

Panne d'un participant

- Dans un fichier Exemple**2a**.java, compléter le protocole pour gérer la panne d'un participant avant de s'être déclaré prêt. Suivre le diagramme de la diapo 13.
  - Le coordinateur décide d'abandonner s'il n'a pas reçu tous les votes dans un délai de 300 unités de temps.
  - Est-ce que le coordinateur attend que les participants en panne aient repris, avant de répondre à l'application ?
  - Que se passe-t-il si un participant qui n'est pas en panne mais trop lent, reçoit la

décision d'abandonner alors qu'il n'a pas encore envoyé son vote au coordinateur ?

- Dans un fichier Exemple**2b**.java, compléter le protocole pour gérer la panne d'un participant **après** s'être déclaré prêt (diapo 14).
  - Le coordinateur répète sa décision à un participant qui la lui demande.

Panne du coordinateur (diapo 15)

- Dans un fichier Exemple**2c**.java, compléter le protocole pour gérer la panne du coordinateur **après** l'étape 1.

Exercice 3 : Gestion des délais

Durée maximum de la validation

- Dans un fichier Exemple**3a**.java, modifier le protocole pour répondre à l'application dans un délai borné . Quelle sont les réponses possibles du coordinateur à l'application, après un temps D, lorsque la validation n'est pas terminée ?

Réponse anticipée

- Dans un fichier Exemple**3b**.java, modifier le protocole pour signaler, à l'application, l'abandon de la transaction dès qu'un participant a voté pour un abandon. Que fait le coordinateur lorsqu'il reçoit le prochain commit en provenance de l'application alors que les participants n'ont pas encore fini la validation précédente.

Exercice 4 : Coordonner plusieurs utilisateurs

- Dans un fichier Exemple**4**.java, modifier le simulateur pour gérer plusieurs transactions provenant simultanément de 2 applications.

Exercice 5: Décision majoritaire

- Dans un fichier Exemple**5**.java, modifier le protocole pour décider d'une validation dès que la majorité des participants a voté pour une validation. Dans ce cas, que répond le coordinateur aux participants minoritaires qui ont voté pour l'abandon ?

Documentation diverse

- tutorial simjava

LesTravauxDirigés, **LesCours**, **Accueil**

Dernière modification le avril 21, 2009 12:20 .
Éditer | Déverrouiller la Page | Supprimer la Page | Historique | Diff | InfosDeLaPage | DebugInfo
Vous êtes connecté en tant que **bdr** | Déconnexion