Université de Lorraine, FST/MIAE, Département Informatique

Examen de l'UE "Architecture et Administration des SGBDR"
Master Informatique, spécialité LOGiciel, Janvier 2012

Durée : 1H40 ; Supports et notes de cours PERSONNELS autorisés

1 Traitement des requêtes

Soit une base de données comportant les schémas de relations :

- **Aéroport** (aéro#, nom-aéro, ville-aéro)
- **Vol** (vol#, cie, périodicité)
- **Liaison** (vol#, tronçon#, aéro-départ#, aéro-arrivée#, heure-départ, heure-arrivée).

Note : La notion de tronçon permet de décrire des vols avec escale et *tronçon#* désigne le numéro de l'escale.

Soit la requête SQL qui permet d'obtenir le numéro de vol et l'heure de départ des vols de la compagnie "AF" partant de Paris-Orly entre 6h et midi :

```
SELECT v.vol#, a.nom-aéro, l.tronçon#, l.heure-départ
FROM   Vol v, Aéroport a, Liaison l
WHERE  a.nom-aéro = "Paris-Orly"
AND    v.vol# = l.vol# AND l.aéro-départ# = a.aéro#
AND    l.tronçon# = 1 AND v.cie = "AF"
AND    l.heure-départ ≤ 12 AND l.heure-départ ≥ 6
```

Optimiser cette requête par la méthode syntaxique (veiller à justifier les différentes étapes de transformation).

2 Traitement des requêtes avec vues

Soient les relations :

- produit(prod#, libelle, pu)
- stock(dep#, prod#, qte)

où prod# désigne un numéro de produit, pu désigne le prix unitaire du produit, dep# désigne un numéro de dépôt de stockage et qte la quantité en stock.
Soit également une vue V définie par :

```
create view V as
      (select libelle, prod# from produit where pu > 100)
```

1. Traiter la requête ci-dessous par la méthode dite syntaxique :

```
select V.prod#, V.libelle, p.pu, s.qte
from produit p, V, stock s
where p.prod# = V.prod# and V.prod# = s.prod#
```

2. Comparer le résultat de la question précédente avec le plan d'exécution ci-dessous engendré par l'optimiseur du SGBD Oracle.

```
-------------------------------------------------------------------
|  ID | parent | OPERATION            | Objet      | OPTIONS    |
- --- - ------ - -------------------- - ---------- - ---------- -
|   0 |        | SELECT STATEMENT     |            |            |
|   1 |      0 | HASH JOIN            |            |            |
|   2 |      1 | HASH JOIN            |            |            |
|   3 |      2 | TABLE ACCESS         | PRODUIT    | FULL       |
|   4 |      2 | TABLE ACCESS         | PRODUIT    | FULL       |
|   5 |      1 | TABLE ACCESS         | STOCK      | FULL       |
-------------------------------------------------------------------
```

3 Jointures et "caching"

L'algorithme de la jointure naturelle ci-dessous est dit algorithme des boucles imbriquées (*nested loops*). $<>$ y dénote la construction de tuples, $||$ la concaténation, *Vide_ens()* l'instanciation d'un ensemble vide et *Ajouter_ens(E, x)* l'ajout d'un élément *x* à un ensemble *E*. Cet algorithme est en $(n \times m)$, n et m étant respectivement le cardinal de R et celui de S.

Début /* $\bowtie (R(X), S(Y), R.A = S.B) = Res(X \cup Y)$ */
$Res = Vide\_ens()$
Pour chaque tuple r de R faire
Pour chaque tuple s de S faire
Si r.A = s.B Alors $Res = Ajouter\_ens(Res, < r||s >)$
Finpour
Finpour
Fin

En considérant un mécanisme d'accès aux données où les tuples sont rangés dans des blocs ou pages, celles-ci étant l'unité de lecture de données par le SGBD, en sachant que *nbExt* et *nbInt* sont, respectivement, les nombres de blocs de stockage de la relations externe (R) et de la relation interne (S) à joindre, comme vu en TD, le coût du nouvel algorithme en nombre d'accès aux blocs est de nbExt*(nbInt +1).

Considérant que *p* soit le nombre de blocs du cache du SGBD, c'est-à-dire l'espace mémoire par lequel transite les n-uplets avant d'être délivrés aux utilisateurs ou aux programmes. En considérant une stratégie qui consiste à lire *(p-1)* blocs d'une relation et *un* bloc de l'autre relation, quelle relation vaut-il mieux utiliser dans la boucle externe : la plus volumineuse ou la moins volumineuse ? Démontrer. **Toute réponse non justifiée sera considérée comme nulle**.