NFP107 et NFP107J
EXAMEN, 1re session – juin 2006
(tout document écrit autorisé)

1 Modèle relationnel (8,5 points)

Un service financier réalise un audit de données bancaires. Le schéma relationnel de ces données est le suivant :

- `compte(idCompte,idClient,solde,resume);`
- `client(idClient,nom,prenom,adresse,annee);`
- `action(idAction,idCompte,montant);`

Les clés primaires sont soulignées. Lorsqu'un même nom d'attribut est utilisé dans plusieurs tables, il s'agit d'une clé étrangère (sauf dans la table où l'attribut est clé primaire, bien entendu). Chaque compte est identifié par un numéro `idCompte`, est en relation avec les clients par `idClient`, est dépositaire d'un `solde` en Euros. L'attribut `resume` contient un résumé des mouvements du compte (un nombre). Chaque client, identifié par un numéro `idClient`, est défini par un `nom`, un `prenom`, une `adresse` et une `annee` de naissance. Chaque action (opération) effectuée sur le compte est identifiée par un numéro `idAction`, est en relation avec un compte par `idCompte`, et porte sur un certain `montant` en Euros. Ce `montant` est positif ou nul dans le cas d'une opération de crédit, et négatif dans le cas d'une opération de débit sur le compte.

Voici un exemple d'instance respectant ce schéma :

compte			
idCompte	idClient	solde	resume
1	30	5	0

action		
idAction	idCompte	montant
1	1	+10
2	1	+5
3	1	-10

client				
idClient	nom	prenom	adresse	annee
30	Dupuis	Jean	1 rue Pasteur	1978

Compréhension du schéma (motivez vos réponses) :

1. (0,5 point) Une compte donné peut-il être affecté à plusieurs clients ?

   Solution : *Non, car il n'y a qu'un seul attribut* `idClient` *pour une valeur de* `idCompte` *donnée, et comme* `idCompte` *est clé primaire, il ne peut y avoir deux lignes dans* `compte` *ayant la même valeur de* `idCompte`.

2. (1 point) Un même client peut-il avoir plusieurs comptes ? Donnez une exemple ou un contre-exemple.

   Solution : *Oui, car* `idClient` *n'a pas de contrainte d'unicité dans* `compte`. *En exemple,* `compte(1,2,_,_)`, `compte(2,2,_,_)`, *et* `client(2,_,_,_,_,_)`.

3. (1 point) Expliquez pourquoi, alors que `(idAction,idCompte)` est clé primaire, un compte peut être en relation avec plusieurs actions.

   Solution : `(idAction,idCompte)` *est clé primaire en tant que paire de valeurs : il ne peut y avoir deux lignes dans la table ayant la même valeur pour la paire. Par contre, une même valeur de* `idAction` *ou de* `idCompte` *peut apparaître plusieurs fois dans la table.*

Donnez les requêtes suivantes, dans le bon langage :

1. (1 point) Le nom et prénom des clients dont le numéro de client est inférieur strictement à 1000 et qui sont nés strictement après 1968 (en SQL et en algèbre) ;

   Solution : *SQL :*

```
select nom, prenom from client
where idClient<1000
and annee > 1968;
```

Algèbre :

$$\Pi_{nom,prenom}(\sigma_{(idClient<1000)\wedge(annee>1968)}(client)).$$

2. (1 point) Les numéros de compte dont le solde est strictement supérieur à 1000 Euros, et dont le propriétaire a pour nom 'Dupont' (en SQL et en algèbre) ;

   Solution : *SQL :*

```
select idCompte from compte, client
where compte.idClient=client.idClient
and compte.solde > 1000 and client.nom='Dupont';
```

   Algèbre :

$$\Pi_{idCompte}(\sigma_{solde>1000}(compte) \bowtie \sigma_{nom='Dupont'}(client)).$$

3. (1 point) Le numéro des clients n'ayant effectué aucun débit (en SQL et en algèbre) ;

   Solution : *SQL :*

```
select idClient from compte C
where not exists
  (select * from action
   where action.idCompte=C.idCompte
   and montant<0);
```

   Algèbre :

$$\Pi_{idClient}(compte) - \Pi_{idClient}(compte \bowtie \sigma_{montant<0}(action)).$$

4. (1 point) Le résumé des mouvements d'un compte est défini de la façon suivante : c'est la *somme des montants de tous les crédits strictement supérieurs à 10 Euros et de tous les débits strictement supérieurs à 50 Euros*. Par exemple, si les opérations sur un compte sont (+50,+20,-10,+5,-60,-20), la somme de contrôle est 50+20-60=10. Donnez tous les numéros d'actions du compte numéro 630 utiles pour les calculs du résumé, *en ordre croissant du numéro d'action* (en SQL uniquement) ;

   Solution : *SQL :*

```
select idAction from action
where idCompte=630 and ((montant > 10) or (montant < -50))
order by idAction;
```

5. (1 point) Donnez, pour chaque compte, son numéro de compte et son résumé (en SQL uniquement) ;

   Solution : *SQL :*

```
select idCompte,sum(montant) from action
where ((montant > 10) or (montant < -50)) ;
group by idCompte;
```

6. (1 point) On suppose que le résultat de la requête précédente est matérialisé dans une table `T(idCompte,resume)`. On soupçonne la table `action` d'avoir été falsifiée par un utilisateur malveillant. Cependant, la table `compte` est intacte. Donnez le numéro des comptes falsifiés (i.e. ceux dont le `resume` du `compte` ne correspond pas au résumé calculé sur `action`).

   Solution :

```
select idCompte from compte,T
and compte.idCompte=T.idCompte
and compte.resume<>T.resume;
```

2 Organisation physique et Optimisation(6,5 points)

La table Client est représentée physiquement par un fichier `FCLIENT` de 1000 pages et un index non dense `Iclient-nom` sur le nom.

1. (1 point) Est-ce que le fichier `FCLIENT` est trié ? Si oui, sur quel(s) attribut(s) ?
   Solution : *oui, sur l'attribut nom.*
2. (1 point) Combien d'entrées a l'index `Iclient-nom`?
   Solution : 1000
3. (0,5 point) Soit la requête

   ```
   select solde from Client,Compte
   where Client.idClient=Compte.idClient
   and nom='Dupont'
   ```

   Que calcule cette requête ?
   Solution : *Le solde des comptes de clients de nom Dupont*
4. (0,5 point) Donner un plan d'exécution logique optimisé pour cette requête sous forme arborescente ou d'expression. relationnelle.
   Solution : *Semi-jointure entre Compte et Client : plus précisément, sélection sur le nom et projection sur idclient de la relation Client suivie de la jointure naturelle avec la relation Compte et projection du résultat sur solde*
5. (1 point) Les seuls index qui existent sont les index sur les clés primaires (`idCompte` pour `Compte` et `idClient` pour `Client`). Par hypothèse, les nombres de blocs (pages) de chaque table, B_{Cl} et B_{Co} sont élevés. Donner un plan d'exécution physique optimisé (arbre ou notation explain).
   Solution :

   ```
   Plan d'execution
   ----------------------------------------------------------
      0      SELECT STATEMENT Optimizer=RULE
      1    0   NESTED LOOPS
      2    1     TABLE ACCESS (FULL) OF 'COMPTE' (TABLE)
      3    1     TABLE ACCESS (BY INDEX ROWID) OF 'CLIENT' (TABLE)
      4    3       INDEX (UNIQUE SCAN) OF 'SYS_C0011154' (INDEX (UNIQUE))
   ```

6. (1,5 points) Expliquer en détail ce plan : dérouler le plan en précisant quelles opérations sont effectuées sur quels objets. Apporter un soin particulier à l'explication et la rédaction.
   Solution : *Jointure par boucles imbriquées et traversée d'index : Accès séquentiel à la table* `Compte` *et projection sur les attributs* `idClient` *et* `solde`*. Pour chaque nuplet obtenu n, traverser l'index de la table* `Client` *sur* `idClient` *avec pour clé $n.idclient$. On obtient un unique rowid de la table* `Client`*. On fait un accès direct à la table* `Client` *et on vérifie si le nom est 'Dupont'. Dans ce cas on garde en sortie le solde et on passe au nuplet n suivant.*
7. (1 point) Soit N le nombre de nuplets de la table `Compte` et I le nombre d'E/S disque pour traverser tout index, donner le nombre d'E/S pour le plan d'exécution ci-dessus. On considère que les nuplets ne se trouvent pas dans les feuilles de l'index.
   Solution : *Le balayage de* `Comptes` *coûte B_{Co} lectures disque. La traversée et l'accès direct pour tester si le nom est 'Dupont' coûte $I + 1$ E/S. Cette étape est faite pour tous les nuplets de* `Compte` *donc N fois. Le nombre total d'E/S est de : $B_{Co} + N(I + 1)$*

3 Concurrence (5 points)

1. Une opération de crédit *Crédit(idCompte, montant)* sur un compte d'identifiant *idCompte* et d'un *montant* donné, doit insérer une nouvelle action dans la table `action`, calculer le nouveau résumé du compte et

le mettre à jour, et aussi modifier le solde du compte. On considère que le calcul du résumé demande seulement des lectures dans la table `action` (parcours de toute la table `action` pour accéder aux actions du compte). On considère que l'insertion d'un nuplet dans une table représente une *écriture de la table*.

(a) (1,5 points) Montrer si transaction **T** : $w[x]r[x]r[y]w[y]c$ peut ou non représenter l'exécution de l'opération *Crédit*, en expliquant la signification de chaque opération et de chaque variable de **T**.

Solution : $w[x]$ *: insertion d'une action (écriture sur la relation* `action`*)*
$r[x]$ *: lecture des actions sur le compte (lecture relation* `action`*) pour calculer la signature*
$r[y]$ *: lecture compte pour récupérer le solde courant*
$w[y]$ *: écriture compte pour modifier le solde et la signature*
$\Rightarrow x$ *= relation* `action`*,* y *= nuplet compte*

(b) (1 point) Montrer qu'une exécution concurrente de plusieurs opérations *Crédit* par verrouillage hiérarchique se fera toujours *en série*, si les verrous d'une transaction sont relâchés après son *Commit*.

Solution : *Une opération Crédit démarre par l'écriture de la relation* `action`*, ce qui bloque toute autre opération Crédit (qui démarre avec écriture sur* `action`*) tant que le verrou d'écriture sur* `action` *est pris, donc jusqu'à la fin de l'opération. Chaque opération Crédit qui obtient ce verrou bloque les autres, ce qui produit une exécution en série.*

2. Soit l'exécution **H** : $w_1[x]r_2[y]r_3[x]r_1[x]w_3[y]r_1[y]c_3w_1[y]c_2c_1$, contenant **T**, mais où toutes les opérations sont considérées au même niveau de granularité.

(a) (1 point) Dire si **H** est sérialisable, en identifiant les conflits et en construisant son graphe de sérialisation.

Solution : *Conflits : sur* x *:* $w_1[x] - r_3[x]$*, sur* y *:* $r_2[y] - w_3[y]$, $r_2[y] - w_1[y]$, $w_3[y] - r_1[y]$, $w_3[y] - w_1[y]$
Le graphe contient un cycle entre T_1 *et* T_3*, donc* ***H*** *n'est pas sérialisable.*

(b) (1,5 points) Trouver l'exécution produite par verrouillage à deux phases simple (pas hiérarchique) sur **H**, si les verrous d'une transaction sont relâchés après son *Commit*. Les opérations bloquées en attente de verrou s'exécutent en priorité quand le verrou devient disponible, en respectant l'ordre de blocage.

Solution : $w_1[x]$ *s'exécute en prenant le verrou d'écriture sur* x
$r_2[y]$ *s'exécute en prenant le verrou de lecture sur* y
$r_3[x]$ *bloquée par* $w_1[x]$*, donc* T_3 *bloquée*
$r_1[x]$ *s'exécute (pas de conflit avec* $w_1[x]$
$w_3[y]$ *bloquée, car* T_3 *bloquée*
$r_1[y]$ *s'exécute, en partageant le verrou de lecture sur* y *avec* $r_2[y]$
c_3 *bloquée, car* T_3 *bloquée*
$w_1[y]$ *bloquée, à cause de* $r_2[y]$*, donc* T_1 *bloquée*
c_2 *s'exécute et relâche les verrous de* $T_2 \Rightarrow w_1[y]$ *s'exécute*
c_1 *s'exécute et relâche les verrous de* $T_1 \Rightarrow r_3[x]$, $w_3[y]$ *et* c_3 *s'exécutent*
Résultat ***H'*** *:* $w_1[x]r_2[y]r_1[x]r_1[y]c_2w_1[y]c_1r_3[x]w_3[y]c_3$