UNIVERSITÉ
FRANÇOIS - RABELAIS
CAMPUS BLOIS

UFR Sciences et Techniques

IUP Blois
Master SIR – 1° année

BD Avancées

TRAVAUX DIRIGÉS

Enseignant **Jean-Yves ANTOINE** (Jean-Yves.Antoine AT univ-tours.fr)

Sécurité des données

CONTRÔLE DES ACCES CONCURRENTS : EXERCICES THEORIQUES

Exercice 1 — Schémas d'exécution sérialisables

Dans cet exerice, on considère deux transactions concurrentes qui veulent mettre à jour une donnée O1. Par exemple :

T1 : UPDATE Table_prix SET prix = prix + 50
T2 : UPDATE Table_prix SET prix = prix * 2

L'exécution de ces deux transactions revient à lire les données PRIX dans un tampon, faire la mise à jour désirée sur ce tampon puis réécrire la valeur obtenue dans PRIX (validation de la transaction).

Problème de concurrence — On suppose qu'il n'existe pas de processus de contrôle des accès concurrents dans notre SGBD. Suivant les circonstances, on peut alors obtenir les deux schémas d'exécution suivants (on ne représente pas les calculs sur les tampons, qui ne concernent pas directement la base de données).

Schéma 1 T1 : read(Prix) ; T2 : read(Prix) ; T1 : write(Prix) + COMMIT ; T2 write(Prix) + COMMIT.

Schéma 2 T1: read(Prix) ; T2: read(Prix) ; T1: write(Prix)+COMMIT ; T2: read(Prix) ; T2: write(T2) + COMMIT

1. Initialement, PRIX vaut 250. Donnez les valeurs obtenues en fin de schéma dans les deux cas. Quel schéma d'exécution pose problème ? Comment a-t-on qualifié ce problème en cours ?
2. A priori, lesquels des deux schémas sont sériels ? sérialisables ? Ce résultat est-il compatibles avec les observations de la question précédente ?
3. Construisez le graphe de précédence des deux schémas. Là encore, retrouve-t-on nos attentes ?

Exercice 2 — Sérialisation par verouilage d'un schéma d'exécution

Dans cet exerice, on considère trois transactions concurrentes qui veulent mettre à jour deux données O1 et O2 :

T0 UPDATE Table_produit_O1 SET nb = 15
UPDATE Table_produit_O2 SET nb = 20

T1 UPDATE Table_produit_O1 SET nb = nb + 1
UPDATE Table_produit_O2 SET nb = nb – 1

T2 UPDATE Table_produit_O1 SET nb = nb + 3
UPDATE Table_produit_O2 SET nb = nb – 3

T3 SELECT nb FROM Table_produit_O1
SELECT nb FROM Table_produit_O2

Les transactions sont lancées dans l'ordre T0, T1, T2 puis T3. On envisage trois schémas d'exécutions

Schéma 1

T0	écriture O1
T0	écriture O2
T1	lecture O1
T1	ecriture O1
T2	lecture O1
T2	ecriture O1
T3	lecture O1
T1	lecture O2
T1	ecriture O2
T2	lecture O2
T2	ecriture O2
T3	lecture O2

Schéma 2

T0	écriture O1
T1	lecture O1
T1	ecriture O1
T0	écriture O2
T2	lecture O1
T2	ecriture O1
T1	lecture O2
T2	lecture O2
T1	ecriture O2
T2	ecriture O2
T3	lecture O1
T3	lecture O2

Schéma 3

T0	écriture O1
T0	écriture O2
T3	lecture O1
T3	lecture O2
T1	lecture O1
T1	ecriture O1
T2	lecture O2
T2	ecriture O2
T1	lecture O2
T1	ecriture O2
T2	lecture O1
T2	ecriture O1

1. Lesquels de ces schémas d'éxécutions sont sériels, sérialisables, sérialisables par permutation ?

2. Donnez les graphes de précédences de ces schémas d'exécution et retrouvez les résultats précédents.

3. Intéressons nous au(x) schéma(s) d'exécution(s) non sérialisable(s). Montrez que ce(s) schéma(s) ne peut être exécuté par une procédure de verrouillage à double phase (2PL) utilisant deux modes : écriture protégée et lecture protégée.

4. On considère toujours ce(s) schéma(s) d'exécution(s). Que se passe-t-il si notre protocole de verrouillage n'est plus 2PL mais utilise des verrous courts.

5. Montrez maintenant qu'un verrouillage 2PL avec verrous en écriture protégée et en lecture protégée n(autorise pas le déroulement de tous les schémas d'exécutions sérialisables.

Exercice 3 — Modes de verrouillages

On considère le schéma d'exécution suivant :

T0	lecture O1
T0	écriture O1
T1	lecture O1
T0	lecture O2
....	

Dans le cadre d'un verrouillage 2PL (verrous longs), précisez si la transaction T1 peut démarrer suivant les modes de verrouillage employés ci-dessous. Justifiez vos réponses.

1. Lecture protégée / Ecriture exclusive
2. Lecture protégée / Ecriture protégée
3. Lecture non protégée / Ecriture exclusive
4. Lecture non protégée / Ecriture protégée
5. Lecture exclusive / Ecriture non protégée

Optimisation des bases de données

INDEXATION : EXERCICES THEORIQUES

Exercice 4 — Indexation par arbre B+ de chaînes de caractères

Comme nous l'avons vu en cours, l'intérêt des index triés hierarchiques à base d'arbre B est d'accélérer l'accès ordonné aux données ou par plage de valeurs. Ce type de requête concerne bien entendu les attributs numériques, mais aussi les données de type chaîne de caractères pour lesquelles un ordre bien connu est l'ordre lexicographique du dictionnaire. Dans cet exercice, nous allons ainsi suivre la construction d'un index organisant dans un arbre B+ des données de ce type chaînes.

1. **Initialisation de l'index** — Pour les besoins de l'exercice, on suppose que l'arbre utilisé est un arbre B+ d'ordre 1 (dans la pratique, l'ordre serait bien entendu bien plus élevé). Sachant que l'arbre doit être équilibré, donné les états sucessifs de ce dernier suite aux ajouts de données suivants: 1) *branche* puis 2) fanal et enfin 3) *arbre*.
2. **Insertion de données dans l'index** — Poursuivez, étape par étape, on considérant maintenant les ajours suivants : 1) *école* puis 2) *drap,* 3) *girafe*, 4) *draperie*, 5) *drapeau*, et enfin 6) *grue*.
3. **Suppression de données dans l'index** — On suppose maintenant que la suppression de tuples dans la base de données entraîne la suppression de certaines valeurs du champ indexé. Donnez les états successifs de l'arbre d'indexation faisant suite aux suppressions suivantes : 1) *branche* puis 2) *fanal* et enfin 3) *drap*.

Exercice 5 — Indexation par tables de hachage : comparaisons

Un autre mode d'indexation fréquemment utilisé par les SGBD est le hachage. Dans cet exercice, nous allons comparer différentes techniques de hachage pour l'indexation de données purement numériques (nombres entiers).

1. **Hachage statique** — Nous avons vu en cours que ce type de hachage à espace d'indexation fixe n'est pas adapté à la problématique des bases de données. Nous allons le vérifier par l'exemple. On considère ici une table de hachage comportant 4 pages de capacité maximale d'indexation de 2 éléments par page (là encore, nous sommes dans un cas jouet bien éloigné des valeurs utilisées en pratique. Deux fonctions de hachage seront considérées en parallèle pour indexer les éléments : dans le premier cas, la fonction de hachage indexera les nombres entiers suivant leur vameur modulo 4. Dans l'autre cas, la fonction retournera la somme des digits du nombre, modulo 4 (par exemple, pour le nombre *456*, on a comme résultat *13 = 4+5+6 modulo 4*).

   a) donnez l'évolution des deux tables de hachage correspondant aux insertions successives des nombres suivants : *16*, *37*, *8*, *2*, *53* et enfin *12*.

   b) combien de valeurs ont-elles pu être indexée avant d'arriver à une saturation de l'index. En comparant ce nombre a la capacité théorique de l'index, pensez-vous qu'une indexation statique aussi simple soit bien équilibrée ?

2. **Hachage dynamique extensible** — Afin de dépasser les limitations du hachage statique, on s'intéresse cette fois à une indexation de type hachage extensible. Comme dans la question précédente, les pages de la table de hachage ne pourront contenir au maximum que 2 éléments indexés. Comme en cours, l'indexation se fera sur les bits de poids faible des nombres indexés, en commençant initialement par ne considérer que le dernier bit des données.

   a) donnez l'évolution de la table de hachage ainsi obtenue suite aux insertions successives des nombres suivants : *16*, *37*, *8*, *2*, *53* et enfin *12*. Cette insertions conduisent-elles à des niveaux de découpage bien équilibrés ?

   b) montrez maintenant comment évolue la table de hachage suites aux suppressions successives suivantes : *37*, *2* et enfin *8*.

3. **Hachage dynamique linéaire** — Pour terminer, nous allons étudier une stratégie de hachage linéaire. Comme dans la question précédente, les pages de la table de hachage ne pourront contenir au maximum que 2 éléments indexés, auxquels on rajoute une page de débordement de 4 éléments. Comme précédemment, l'indexation se fera sur les bits de poids faible des nombres indexés suivant le niveau de découpage atteint par le pointeur dynamique de la table.

   a) donnez l'évolution de la table de hachage ainsi obtenue suite aux insertions successives des nombres suivants : *16*, *37*, *8*, *2*, *53* et enfin *12*. Tous les éléments sont-ils bien indexés à la fin de ces différentes insertions.

b) on ajoute enfin le nombre *1* dans l'index. Que se passe-t-il pour le nombre présent dans la page de débordement ? Que devient par contre le nombre *1*. L'index obtenu est-il plus équilibré que l'index extensible équivalent ?

Exercice 6 — Optimisation de bases de données (examen 2004-2005 - 3 points)

On considère un attribut d'une table d'une base de données qui prend pour occurrences à un instant donné les valeurs suivantes : {1,2,5,8,9,12,14}. Afin d'optimiser l'accès aux données, on désire indexer cet attribut sous la forme d'un index hiérarchique correspondant à un arbre B+ d'ordre 1.

1 — Donnez l'arbre B+ correspondant à l'indexation de l'ensemble de valeurs donné ci-dessus. Peut-il y avoir plusieurs arbres possibles ?

2 — Donnez le nouvel arbre B+ obtenu après insertion dans la base de données de la valeur 3 ?

3 — Donnez le nouvel arbre B+ obtenu après insertion dans la base de données de la valeur 4 ?

4 — Pourquoi la définition d'un index hiérarchique n'est-elle pas efficace si l'attribut a peu de valeurs distinctes ? Solution ?

5 — Pourquoi la définition d'un index sur un attribut n'est-elle pas efficace si la table contient peu d'enregistrements ?

Exercice 7 — Optimisation de bases de données (examen 2006-2007 - 3 points)

1 — Elisa souhaite indexer un champ *code_postal*, sachant que notre objectif est de l'utiliser, par exemple, pour rechercher toutes les entreprises d'un département (par exemple, si on cherche les entreprises du Loire-et-Cher, on cherchera uniquement les valeurs indexées comprises entre 41000 et 41999). Quel type d'index conseillez-vous d'adopter pour cela ? Justifiez votre réponse.

2 — Pouvez-vous donner les états successifs de cet index par ajout (en partant de rien) des codes postaux 41500 puis 41000, 41800, 41600, 41900, 41700 et enfin 41200 (**Attention** : cette question ne sera considérée que si vous avez choisi le bon type d'index).

PL/SQL

Exercice 8 — Programmation de blocs PL/SQL (adapté de Soutou, 2005)

On considère une base de données permettant la gestion d'un parc informatique. Elle se compose des tables suivantes :

- `salle` décrit l'équipement de chaque salle de TP
- `poste` décrit chaque poste de TP installé
- `logiciel` décrit chaque logiciel installé sur au moins une machine de l'IUP
- `install` décrit l'ensemble des installations de logiciels faites sur les machines de l'IUP

Les tableaux suivants donnent le schéma de chacun de ces quatre tables (attributs).

SALLE	**Description**
no_salle	Numéro de la salle (clé primaire) – entier
nom_salle	Nom de la salle – chaîne de 50 caractères maximum – renseigné
nb_postes	Nombre de postes dans la salle – entier renseigné

POSTE	**Description**
no_poste	Numéro du poste (clé primaire) – entier
no_salle	Référence au numéro de la salle où est installé le poste
type_OS	Système d'exploitation installé – chaîne de caractères prenant les valeurs *Linux*, *XP* ou *Vista*
IP	Adresse IP du poste – chaîne de 15 caractères

LOGICIEL	**Description**
no_soft	Numéro de référence du logiciel concerné (clé primaire) – entier
nom_soft	Nom du logiciel – chaîne de 50 caractères renseignée
editeur	Nom de l'éditeur du logiciel – chaîne de 50 caractères
version	Numéro de version du logiciel – chaîne de 5 caractères renseignée
type_OS	OS sur lequel fonctionne le logiciel – chaîne de caractères prenant les valeurs *Linux*, *XP* ou *Vista*
date_achat	Date d'achat (ou de récupération pour un gratuiciel) – type DATE renseigné
date_exp	Date éventuelle d'expiration de la licence – type DATE

INSTALLATION	**Description**
no_soft	Référence au logiciel concerné – entier
no_poste	Référence au poste concerné – entier
date_install	Date d'installation du logiciel sur ce poste – type DATE renseigné

1 — Donnez les requêtes SQL permettant la création de la base de données.

2 — Donnez quelques requêtes SQL qui permettent d'insérer quelques tuples dans chaque table de la base de données.

3. Bloc PL/SQL et variables %TYPE — Ecrire un bloc PL/SQL qui affiche les détails de la dernière installation effectuée. Ce bloc PL/SQL devra permettre une maintenance la plus aisée possible suite à des changements de type dans la base de données. Exemple d'affichage souhaité en sortie :

```
Derniere installation effectuee
-----------------------------------
Salle = TP05
Poste = TP05_014
Logiciel = Gimp
Date installation = 01/03/2007
```

4. Variables de substitution et globales — Ecrire un bloc PL/SQL qui saisit un numéro de salle et un type de poste, et

qui retourne un message indiquant le nombre de postes et d'installations correspondantes sous le format suivant :

```
Salle = TP05
Type poste = XP
-------------------------------------
G_NBPOSTE   = 14
G_NBINSTALL = 88
```

Vous utiliserez pour la saisie des variables de substitution et pour les résultats des variables globales (`G_NBINSTALL` et `G_NBPOSTE`).

Remarque : si vous testez votre programme en TP, pensez à exécuter votre bloc avec la commande `start` déjà vue, et non pas par copier-coller dans l'interface Oracle, ceci à cause des ordres `ACCEPT`.

5. Une transaction en PL/SQL — Ecrire une transaction qui permette d'insérer un nouveau logiciel dans la base après avoir saisi toutes ses caractéristiques, la date d'achat étant celle du jour. La trâce ci-dessous vous donne un exemple d'exécution du bloc PL/SQL correspondant.

```
SQL > start bloc_insertion
Numéro du logiciel = log20070314
Nom du logiciel = Sqlloader
Editeur = Oracle
Version du logiciel = 10i.4
Systeme d'exploitation = Linux
Date d'achat = 10-03-2007 13:48:23
Date d'expiration = 10-03-2008
```

6. Curseur — On désire connaître, pour chaque logiciel installé, le temps passé (en nombre de jours) entre l'achat et l'installation du logiciel sur chaque poste.

6.1. Donnez la commande LDD qui modifie la table *installation* pour rajouter un attribut *delai*, de type entier, dans lequel sera conservée cette information. On remarquera qu'il n'est pas possible de poser de contrainte d'intégrité imposant que la date d'installation soit postérieure à la date d'achat du logiciel.

6.2. Ecrire un bloc PL/SQL qui renseigne cette information pour toute la base de données et affiche également une trâce de résultat en sortie. On gérera au passage les situations incohérentes où la date d'installation est antérieure à la date d'achat (attribut laissé à NULL et message d'information à l'exécution) :

```
Logiciel Gimp version 2.2 installe sur poste TP04_012. Delai : 34 jours
Logiciel Oracle version 10i intalle sur poste TP05_035. DATE ERRONEE
```

7. Une procédure d'installation — En pratique, il est fastideux de rentrer un nouveau tuple dans la base installation après installation. Pour faciliter cette tâche, écrire une procédure *install_salle* permettant d'enregistrer l'installation groupée d'un nouveau logiciel sur tous les postes d'une salle particulière. On suppose que le logiciel concerné a déjà été entré dans la table *logiciel*. L'installation se fera à la date du jour maintenue par le SGBD. En paramètres, on passera le numéro de la salle concernée, le nom et la version du logiciel. Le programme informera l'utilisateur de son bon déroulement par une trâce explicite :

```
SQL> EXECUTE install_salle('TP04','Oracle','10i')
```

7.1. Ecrire la procédure PL/SQL demandée, en ne tenant pas compte des éventuelles exceptions.

7.2. Exceptions — Modifier la procédure pour tenir compte des exceptions les plus probables (logiciel ou salle inconnus, système d'exploitation incompatible avec le logiciel dans cette salle...). La gestion de ces exceptions se limitera à l'affichage d'un message d'erreur.

7. Déclencheur — Pour terminer, il est demander de programmer les déclencheurs suivants :

8.1. Mise à jour d'attribut — Ecrire le déclencheur *Trig_MAJ_nb_postes* qui réalise la mise à jour automatique de l'attribut *nb_postes* de la table *salle* après tout ajout ou suppression de poste dans une salle.

8.2. Programmation de contraintes — Nous avons vu à la question 6.2.qu'il n'était pas possible de définir directement dans la base une contrainte d'intégrité qui impose que la date d'installation d'un logiciel soit postérieure à sa date d'achat. Réaliser un déclencheur qui programme précisément cette contrainte.

