

Université Paris 7  
Master 1 Informatique, Bases de données avancées.

11 janvier 2010

Durée : 3 heures. Documents manuscrits, notes de cours, notes de TD/TP autorisés. Livres interdits.

Le sujet comporte 3 pages.

## Dépendances

**Exercice 1** On considère une relation  $R(A, B, C, D, E)$  avec les dépendances fonctionnelles suivantes :

$$\begin{aligned}A, B &\rightarrow C \\ C, D &\rightarrow E \\ D, E &\rightarrow B\end{aligned}$$

**Question 1:** Est-ce que  $AB$  est une clé candidate ? Justifier.

**Question 2:** Est-ce que  $ABD$  est une clé candidate ? Justifier.

**Question 3:** Trouver toutes les clés candidates.

**Question 4:** Est-ce que cette relation est en 3NF ? Est-ce qu'elle est BCNF ? La réponse doit être justifiée.

**Exercice 2** Nous avons une relation

$\text{Livre}(\text{Titre}, \text{Auteur}, \text{Editeur}, \text{Type\_Livre}, \text{Prix}, \text{Affil\_Auteur})$

avec les dépendances fonctionnelles :

$\text{Titre} \rightarrow \text{Editeur}, \text{Type\_Livre}$

$\text{Auteur} \rightarrow \text{Affil\_Auteur}$

**Question 5:** Montrer que  $\text{Titre}, \text{Auteur}$  est une clé candidate.

**Question 6:** Expliquer (sans faire des calcul) pourquoi il n'y a pas d'autres clés candidates.

**Question 7:** Est-ce que cette relation est en 3NF ? en BCNF ? Justifier la réponse.

**Question 8:** Décomposer la relation  $\text{Livre}$  en relations BCNF.  
Votre décomposition préserve-t-elle les dépendances fonctionnelles ?

## Modélisation

**Exercice 3** Le but de cet exercice est de concevoir une base de données de gestion de notes des élèves d'une école. Les entités possibles : élève, professeur, matière, note, etc. (vous pouvez choisir d'autres entités, c'est juste un exemple).

Bien sûr votre BD devait permettre de retrouver les informations habituelles, sans doute il est intéressant de savoir, par exemple, qui a obtenu une note donnée et dans quelle matière.

**Question 1:** Proposez un modèle UML pour cette BD.

Le type de chaque relation (un à plusieurs, plusieurs à plusieurs, un à un) doit être clairement visible sur le dessin.

**Question 2:** En déduire le schéma de la BD, c'est-à-dire quelles tables obtenez-vous à partir de votre modèle ?

On ne demande pas d'écrire les commandes CREATE TABLE pour les tables mais vous devez indiquer pour chaque table qui résulte de votre modèle :

- le nom de la table,
- les attributs,
- la clé et
- les clés étrangères (s'il y en a).

**Remarque :** C'est un exercice qui apporte des points s'il est bien fait. Par contre si mal fait cela ne rapporte pas grand chose.

En tout cas il vaut mieux bien modéliser sans faire de tables que mal modéliser et faire mal les tables par la suite (moins mais bien fait est préférable que plus mais mal fait).

## Triggers - déclencheurs

### Exercice 4

Une base de données d'un entrepôt contient (entre autre) deux tables

```
Stock(id_produit,quantite)
Commande(id_commande,id_produit,id_client,date_commande,date_envoie,qty)
```

La table Stock contient pour chaque produit (identifié par id\_produit de type INT) la quantité en stock de type INT. La quantité en stock est toujours  $\geq 0$ .

La table Commande sert à recenser les commandes. Chaque commande est identifiée par id\_commande unique de type INT. Les attributs id\_produit, id\_client sont de type INT également.

date\_commande et date\_envoie sont de type DATE. La valeur de date\_envoie est NULL tant que la commande n'est pas expédiée. Par contre date\_commande n'est jamais NULL.

qty de type INT, toujours positif, donne la quantité commandée.

Il faut écrire les triggers suivants (pour chaque trigger il faut donner la fonction pgsql qui l'implémente et aussi la commande CREATE TRIGGER associée) :

**Question 1:** Le premier trigger sera associé à la table Commande et sera déclenché au moment d'insertion d'une nouvelle commande (c'est-à-dire de voir ce qui convient plus : après ou avant l'insertion). Le trigger vérifiera si pour le produit commandé la quantité commandée qty est inférieure ou égale à la quantité en stock.

- 
- Si Stock.quantite pour le produit commandé est déjà 0 alors la commande sera refusée.
- 

c'est-à-dire la commande ne sera pas ajoutée dans la table Commande.

- Si pour le produit commandé  $\text{Commande.qty} \leq \text{Stock.quantite}$  c'est-à-dire le stock est suffisant pour réaliser complètement la commande alors la commande doit être ajoutée dans la table de commandes et on diminuera la quantité dans la table Stock en soustrayant la quantité commandée qty.
- Si le stock est supérieur à 0 mais pas suffisant pour respecter complètement la commande alors on enregistre la commande dans la table Commande mais avec la valeur qty égal à la quantité en stock et on met 0 dans la valeur quantite de la table Stock pour le produit commandé.

**Question 2:** Le deuxième trigger sera associé à la table `Commande` et sera déclenché à la suppression d'une commande. Le trigger doit empêcher la suppression d'une commande déjà envoyée, c'est-à-dire avec `date_envoie` non NULL.

Par contre si `date_envoie` est NULL alors la suppression sera effectuée mais le trigger doit en plus rajouter la quantité `qty` de la commande supprimée dans la quantité en stock de la table `Stock` pour le produit dont la commande est supprimée.

## Transactions

**Exercice 5** Nous avons trois transactions  $T_1, T_2, T_3$  exécutée simultanément. On considère deux histoires possibles :

$H_1 : r_1(X), r_2(Z), r_1(Z), r_3(X), r_3(Y), w_1(X), w_3(Y), r_2(Y), w_2(Z), w_2(Y)$

et

$H_2 : r_1(X), r_2(Z), r_3(X), r_1(Z), r_2(Y), r_3(Y), w_1(X), w_2(Z), w_3(Y), w_2(Y)$

$r_i(A)$  désigne une lecture d'une variable  $A$  par la transaction  $i$ ,  $w_i(A)$  désigne l'écriture.

Pour chaque histoire vérifier si elle est sérializable (la réponse doit être justifiée) et si c'est le cas donner une histoire sériale équivalente.

**Exercice 6** Nous avons une transactions  $T_1$  suivante :

$T_1 : r_1(X), w_1(Y), w_1(X), r_1(X), r_1(Y)$

Le système de gestion de transactions ajoute dedans les opérations de verrouillage et déverrouillage.

On suppose que les opérations suivantes sont implémentées :

- `read_lock(X)` - poser le verrou de lecture (partagé),
- `write_lock(X)` - poser le verrou d'écriture (exclusif),
- `read_unlock(X)` et `write_unlock(X)` - lever le verrou,
- `upgrade(X)` - transforme le verrou de lecture en verrou d'écriture,
- `downgrade(X)` - transforme le verrou d'écriture en verrou de lecture.

Le système de gestion de transaction utilise l'algorithme de verrouillage en deux phases mais pour augmenter le parallélisme on suppose qu'il respecte les consignes suivantes :

- (A) chaque verrou est posé le plus tard possible et lever le plus tôt possible,
- (B) on préfère de poser les verrous de lecture que les verrous d'écriture si cela est possible et si cela suffit.

Ajouter les opération de verrouillage/déverrouillage dans  $T_1$  en respectant les consignes ci