# TD 4 : Intégrité, vues et confidentialité

# Corrigé

**1. Contraintes d'intégrité**

a) ALTER TABLE Lecteur ADD ( CONSTRAINT domaine_conso CHECK (Consommation IN ('Forte', 'Moyenne', 'Faible')));

b) ALTER TABLE Emprunt ADD ( CONSTRAINT fk_lecteur FOREYGN KEY NumLecteur REFERENCES Lecteur); ALTER TABLE Emprunt ADD ( CONSTRAINT fk_livre FOREYGN KEY Cote REFERENCES Livre);

c) ALTER TABLE Livre ADD ( CONSTRAINT auteur_catégorie CHECK (NOT EXISTS (SELECT Catégorie FROM Livre GROUP BY Auteur HAVING COUNT(Catégorie) > 1)));

**2. La gestion des vues**

**Question 1**: Définition d'une vue

**a) Définition de la vue POLICIER**

```
create view POLICIER
as     select Cote, Titre, Auteur
 from LIVRE
 where Catégorie = "roman policier";
```

Cette vue est un exemple d'une définition de schéma externe où le sous-ensemble de la base de données est défini comme un sous-ensemble au niveau des occurrences.

**b) Définition de la vue LECTEUR-DE-POLICIER**

```
create view LECTEUR-DE-POLICIER
as     select Num,Nom,Adresse,Cote,
       DateEmprunt, DateRetour
 from LIVRE L, LECTEUR E, PRET P
 where L.Cote = P.Cote
       and E.Numéro = P.Numlecteur
       and L.Catégorie = "roman policier";
```

Cette définition de vue est un exemple de restructuration de l'information. L'utilisateur ne voit dans la vue qu'une seule relation alors qu'en réalité la base est composée de trois relations. Cette possibilité permet de simplifier le travail des utilisateurs en leur présentant les informations sous le format qui leur convient le mieux, en leur évitant par exemple la formulation de jointures lors d'interrogations de la vue.

**c) Définition de la vue LECTEUR-INTERDIT**

```
create view LECTEUR-INTERDIT (Numéro,Nom)
as      select Numéro, Nom
 from LECTEUR E, PRET P
 where E.Numéro = P.Numlecteur
       and (DateEmprunt + 15) < DateSystème;
```

On suppose que le système permet d'obtenir la date du jour en utilisant le mot-clé "date-système". La définition de la vue est dynamique, en ce sens que son contenu varie tous les jours automatiquement en fonction de la date. Cette définition de vue permet également de changer les noms des attributs pour s'adapter aux habitudes des utilisateurs.

**d) Définition de la vue STATS-DES-PRETS**

```
create view STATS-DES-PRETS(NumLecteur,NombrePrêts)
as      select Numéro, count(*)
 from LECTEUR E, PRET P
 where E.Numéro = P.Numlecteur
       and DateEmprunt > "d"
 group by Numlecteur;
```

L'attribut NombrePrêts est une information déduite dans la mesure où elle n'est pas stockée en tant que telle dans la base de données, mais calculée à partir des informations stockées.

**Question 2 :** Utilisation d'une vue

**a) La question utilisateur est**

```
select distinct Nom
from LECTEUR-DE-POLICIER
where DateEmprunt = "d";
```

La même question posée sur les relations de la base serait:

```
select Nom
from LIVRE L, LECTEUR E, PRET P
where L.Cote = P.Cote
 and E.Numméro=P.Numlecteur
 and L.Catégorie = "roman policier"
 and DateEmprunt = "d";
```

Cette question est beaucoup plus complexe à formuler car elle comporte deux jointures, alors que la question sur la vue est très simple puisqu'elle ne comporte qu'une restriction. C'est l'un des avantages des vues: simplifier le travail des utilisateurs.

**b) La question de l'utilisateur exprimée en SQL est la suivante:**

```
select distinct Nom
from LECTEUR-DE-POLICIER E, POLICIER P
where E.Cote = P.Cote and Auteur = "SIMENON"
group by NumLecteur, DateEmprunt
having count (Numlivre)>3;
```

La question telle qu'elle aurait dû être posée sur la base de données est la suivante:

```
select distinct Nom
from LECTEUR E, LIVRE L, PRET P
where P.Cote=L.Cote and Auteur="Simenon"

  and L.Catégorie="roman policier"

  and P.Numlecteur=E.Numéro
group by Numlecteur,DateEmprunt
having count (Numlivre) >3;
```

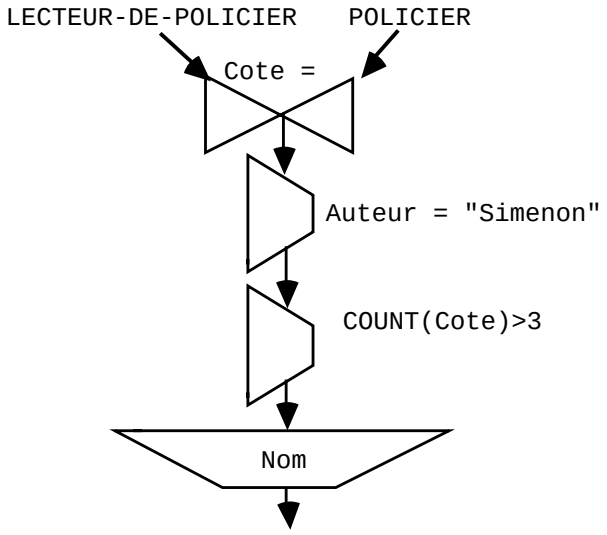
**Question 3:** Traitement d'une question portant sur une vue

L'algorithme de transformation d'une question posée sur une ou plusieurs vues consiste à "additionner" la question utilisateur et les questions définissant chacune des vues. Le traitement est itératif car une vue peut être construite sur d'autres vues.

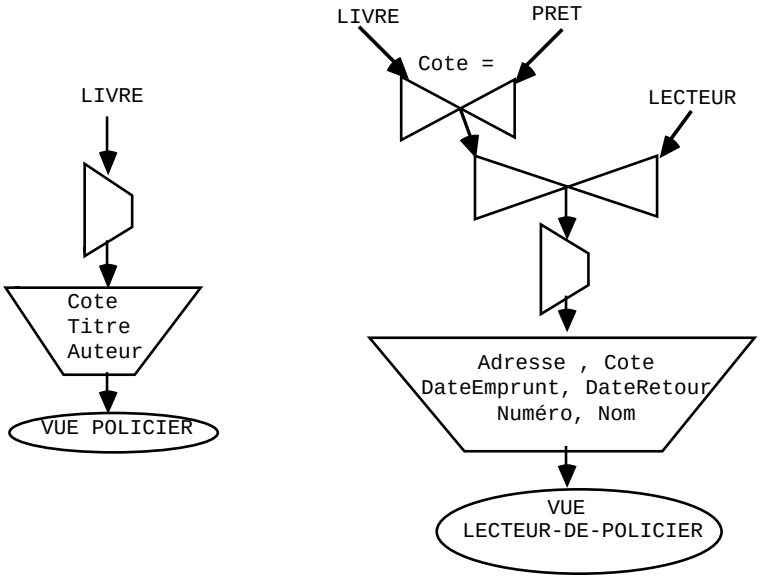
```
Tant que la question comporte l'utilisation de vues
•  rechercher dans la métabase les arbres de définition des vues
•  construire l'arbre syntaxique de la question posée
•  changer les noms des attributs de la vue dans les noms
   d'attributs correspondants s'il y a re-nomination des attributs
   dans la vue
•  transformer les arbres en un seul en faisant coïncider le
   résultat d'une vue avec le nom de la vue dans la question
•  simplifier éventuellement l'arbre en éliminant les opérations
   redondantes
ftq
Passer l'arbre transformé de la question à l'optimiseur de question
```

Application de l'algorithme sur l'exemple :

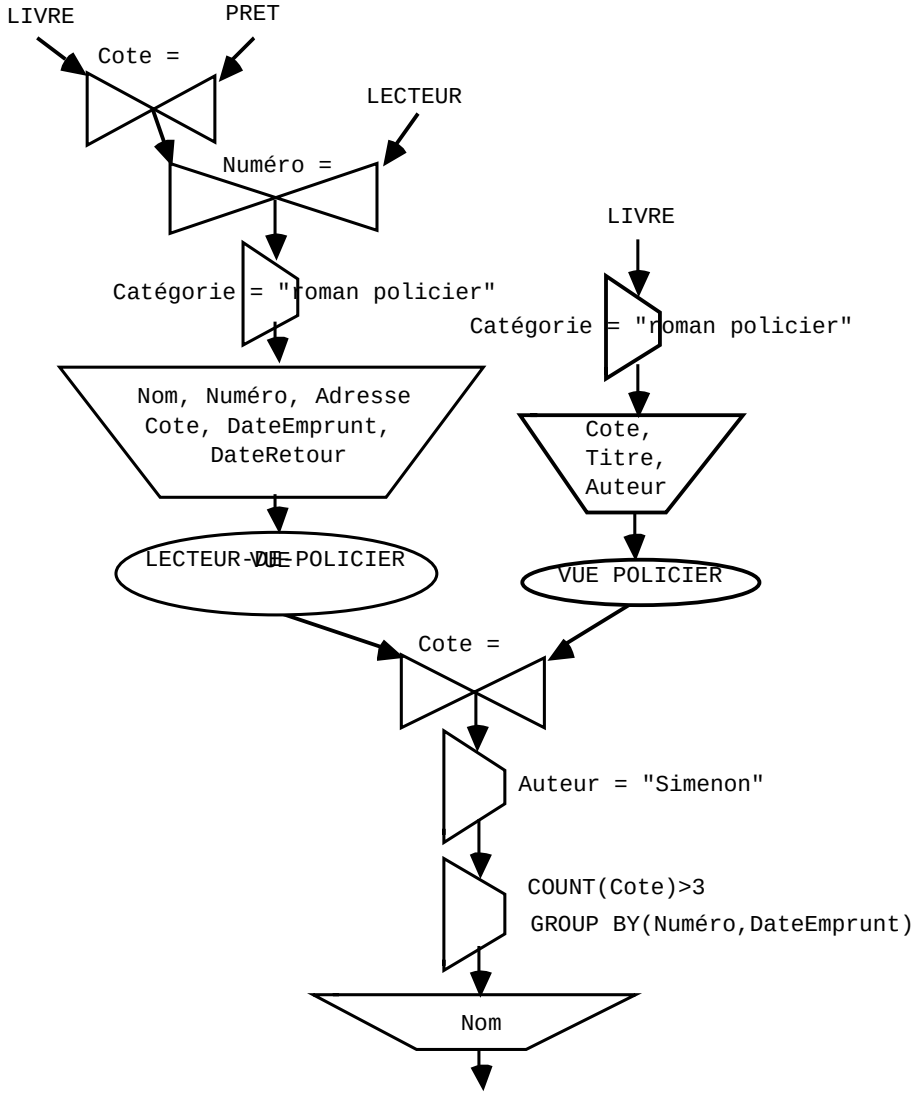
1) L'arbre de la question est le suivant:




2) Les arbres des deux vues sont les suivants:



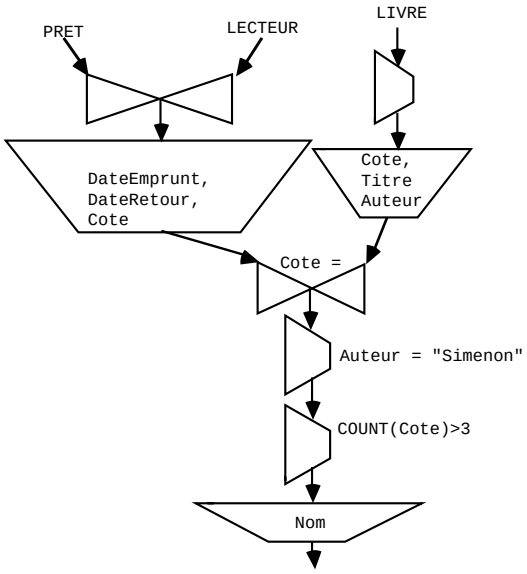

3) Ajout des arbres en faisant coïncider le résultat d'une vue et le nom de la vue:



4) Dans la définition des vues il n'y a pas de renomination des attributs.

5) Simplification de l'arbre.

On utilise deux fois la relation LIVRE pour une jointure sur n°livre. Il y a également deux restrictions identiques avec le critère <catégorie = "roman policier">. On ne garde qu'une fois chacune de ces opérations.




**Question 4**: Mise à jour à travers une vue

La mise à jour à travers les vues est interdite dans le cas général car elle pose différents problèmes :

(1) Attributs non définis dans la vue: toute insertion d'un tuple dans la vue entraîne l'insertion d'un tuple dans la base ayant une valeur indéterminée pour les attributs non visibles dans la vue. Par exemple l'insertion d'un nouveau tuple dans la vue LECTEUR-INTERDIT implique l'insertion d'un tuple LECTEUR dont l'adresse est inconnue. Toute vue incluant dans sa définition une projection entraîne ce type de problème.

(2) Risques d'incohérence quand une relation d'une vue est obtenue par jointure de plusieurs relations réelles. Par exemple, l'insertion d'un tuple dans LECTEUR-DE-POLICIER signifie-t-elle l'ajout d'un prêt pour un lecteur existant, ou l'insertion d'un prêt et d'un nouveau lecteur, ou encore l'insertion d'un prêt, d'un livre et d'un lecteur? L'ambiguïté est la même pour les suppressions de tuples. La répercussion d'une mise à jour dépend de la sémantique de la vue et de l'application. Elle ne peut pas être automatique. Il faudrait décrire pour chaque vue la sémantique des opérations de mises à jour.

(3) Perte de signification de la mise à jour: la suppression, par exemple, du tuple (1347, 45) de la vue STATS-DES-PRETS, ne peut pas être répercutée au niveau des relations de la base. Cette perte de signification est caractéristique des vues construites en utilisant des agrégats et/ou des fonctions de calcul somme ou moyenne.

En général les SGBD interdisent les mises à jour à travers les vues. Certains les autorisent dans le cas où la sémantique des mises à jour peut être définie sans ambiguïté. C'est le cas si la définition de la vue n'utilise que l'opérateur de restriction d'une relation (ni jointure, ni projection, ni agrégat, ni fonction de calcul). Dans ce cas un tuple de la vue correspond à un seul tuple de la base. On peut autoriser également l'opérateur de projection si le SGBD gère les valeurs indéterminées.

**Question 5**: Vues concrètes ou clichés

Le temps de calcul et de stockage d'un cliché peut être important si le sous-ensemble de la base de données correspondant est volumineux. Pour réduire ce temps d'exécution, qui pour un cliché est répétitif, le système peut utiliser un mécanisme de mise à jour différentiel. Seuls les tuples insérés, supprimés ou modifiés dans la base depuis le dernier calcul sont à leur tour insérés, supprimés ou modifiés dans le cliché. Il est nécessaire que le SGBD garde les tuples insérés ou supprimés des relations de la base dans une relation spéciale. Les suppressions de tuples posent un problème particulier dans le cas général: un tuple du cliché peut provenir de plusieurs tuples de la base et la suppression d'un seul de ces tuples ne doit pas entraîner la suppression du tuple du cliché. Ce problème est étudié dans [Kerhervé 86]. L'idée est d'associer à chaque tuple de la vue un compteur indiquant le nombre de raisons pour sa présence. Lors d'une suppression d'un tuple de la base le compteur correspondant est décrémenté. Quand le compteur devient nul, le tuple du cliché est supprimé.

### 3. La gestion des autorisations

**Question 1**: Exemple d'utilisation

```
grant select insert on LIVRE
to dupont
with grant option
```

**Question 2**: Vérification de l'attribution des droits.

Une requête GRANT a les paramètres suivants:

1. l'usager ayant émis le GRANT, appelé donateur;
2. l'usager à qui les droits sont accordés, appelé bénéficiaire;
3. la liste des droits accordés (un droit étant accordé avec ou sans droit de transmission), chacune des deux possibilités est identifiée comme un droit différent;
4. la relation sur laquelle les droits sont accordés appelée objet.

La relation DROIT permet de mémoriser ces informations. Son schéma peut être le suivant:

```
DROIT(relation-objet,nom-bénéficiaire,nom-donateur,
 droits-non-transmissibles,droits-transmissibles)
```

Ce schéma n'est pas en première forme normale car les attributs droits-transmissibles et droits-non-transmissibles contiennent plusieurs valeurs, mais il permet un stockage plus compact. Il nécessite inversement un traitement pour analyser les droits présents.

L'algorithme de GRANT comprend les étapes suivantes:

```
1) vérifier que le donateur possède les droits qu'il veut accorder
   avec le droit de transmission.
   Il faut pour cela extraire de la relation DROIT les droits que
   l'auteur du GRANT possède sur la relation 'objet'.
   select droits-transmissibles
   from droit
   where nom-bénéficiaire = "donateur"
     and relation-objet = "objet";
2) si le donateur possède effectivement le droit de transmettre
   'liste-droits', ces derniers doivent être mémorisés dans la
   relation DROIT par une requête INSERT, soit comme droits
   transmissibles soit comme droits non transmissibles suivant la
   présence ou non de la clause with grant option.
   L'algorithme de grant générera la requête suivante:
   insert into DROIT
   values("objet","bénéficiaire","donateur",nil,
                                        "liste-droits")
   ou
   insert into DROIT
   values("objet","bénéficiaire","donateur",
                                        "liste-droits",nil)
```

**Question 3**: Vérification du retrait d'un droit

Dans un système à gestion des droits décentralisée où les droits sont accordés, vérifiés et retirés dynamiquement, le contrôle du retrait des droits n'est pas trivial. La vérification de la provenance des droits d'un usager nécessite en principe de tracer le graphe de transmission des autorisations pour vérifier la provenance de chaque droit et s'assurer qu'un usager ne s'est pas retransmis un droit à lui-même par une voie détournée. Une solution plus simple que la construction du graphe a été développée dans [ Lindsay79]. On associe à chaque droit une estampille indiquant la date à laquelle le droit a été accordé. L'estampille permet de reconnaître les droits qui restent à un usager X, quand un droit qui lui avait été accordé à une date *t* disparaît. Tous les droits issus récursivement de X et postérieurs à *t* doivent également disparaître. Le format de la relation `DROIT` devient le suivant:

```
DROIT ( relation-objet,
  nom-bénéficiaire,
  nom-donateur,
  droits-non-transmissibles,
  estampilles-droits-transmissibles )
```

Chaque droit transmissible est représenté, non plus par un bit mais par l'heure à laquelle il a été accordé. Un droit non accordé a une estampille 0. Un exemple du contenu de la relation (en ne considérant que les droits transmissibles) est le suivant:

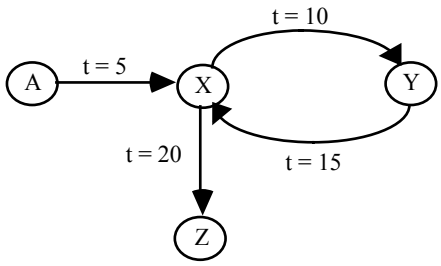
| relation objet | nom bénéficiaire | nom donateur | select | insert | delete | update |
|---|---|---|---|---|---|---|
| LIVRE | X | A | 15 | 15 | 0 | 0 |
| LIVRE | X | B | 20 | 0 | 20 | 0 |
| LIVRE | Y | X | 25 | 25 | 25 | 0 |
| LIVRE | X | C | 30 | 0 | 30 | 0 |

Supposons qu'à l'instant t = 35, B retire tous les droits qu'il avait accordés à X au temps t = 20. Le tuple (LIVRE, X, B, 20, 0, 20, 0) doit disparaître. Les droits qui restent à X sont alors (LIVRE, X, (A,C), (15, 30), 15, 30, 0). Les droits que X a transmis sont (LIVRE, Y, X, 25, 25, 25, 0). Le droit DELETE donné par X au temps t = 25 doit disparaître puisque l'on considère à présent que X a reçu ce droit au temps t = 30. En revanche les droits SELECT et INSERT accordés à Y au temps 25 sont conservés car ils proviennent de droits reçus par X avant le temps 20. L'algorithme est le suivant:

```
proc REVOKE (b: nom-bénéficiaire, da: droit-accès, r: relation-
objet, d: nom-donateur)
/* retirer le privilège 'da' accordé par'd'à'b' sur 'r' */
 update DROIT
 set da = 0
 where nom-bénéficiare = b
       and nom-donateur = d
       and relation-objet = r;
/* définir l'estampille minimum qui reste à 'b' pour le droit
'da' sur la relation 'r' */
 select MIN(da) into min
 from DROIT
 where nom-bénéficiare = b
       and relation-objet = r;
/* retirer tous les droits 'da' accordés par 'b' sur 'r' avant
'min' */
 select nom-bénéficiaire into liste-retrait
 from DROIT
 where nom-donateur='b'
       and relation-objet='r'
       and da<'min';
 Si liste-retrait • vide
 alors pour chaque élément 'l' de liste-retrait
        faire
             REVOKE (l, da, r, b)
        fpour
 fsi
fin proc
```

L'algorithme est récursif et retire de proche en proche les droits à tous ceux auxquels il ne peut plus avoir été accordé.

Exemple :




Si A retire son droit à X, il faut récursivement l'enlever à tous les autres. L'algorithme se déroule ainsi:

- 1ère passe : REVOKE( X,da,r,A)
  retire à X le droit 'da' qui lui vient de A
  minimum des estampilles qui restent à X: 15
  ==> retirer le droit à Y donné en t = 10 (donc avant 15) :
  REVOKE (Y, da, r, Y)
- 2ème passe : REVOKE (Y, da, r, Y)
  Y n' a plus le droit 'da', donc il faut retirer ce droit à ceux auxquels il l'a transmis:
  REVOKE (X, da, r, Y)
- 3ème passe: REVOKE (X, da, r, Y)
  X n'a plus le droit 'da', il faut donc le retirer à ceux auxquels il l'a transmis:
  REVOKE (Z, da, r, X)

**Question 4**: Vérification des droits d'accès lors des manipulations

Lors de chaque requête d'un utilisateur le système vérifie que cet usager a effectivement le droit d'exécuter ce type de requête sur les relations citées. Cette vérification pourrait se faire par un accès à la relation `DROIT`, mais cet accès génère alors une entrée-sortie par requête ce qui est lourd. Une meilleure solution consiste à n'accéder à la relation `DROIT` qu'une fois en début de session pour chaque utilisateur. Quand un utilisateur se connecte au SGBD, après son authentification (vérification qu'il a bien le droit d'utiliser le SGBD), le système lit en mémoire centrale la partie de la relation `DROIT` qui le concerne. Cet extrait de la relation `DROIT` pour un usager 'U' est défini par:

```
select * into DROITS-MC
from DROIT
where nom-bénéficiaire = 'U';
```

La relation `DROITS-MC` doit être stockée en mémoire centrale dans un emplacement réservé au système auquel un utilisateur ne peut jamais avoir accès pour éviter toute modification frauduleuse ou accidentelle de cette relation.

**Question 5**: Droits d'accès sur une vue

Quand un utilisateur crée une vue, il ne peut obtenir sur celle-ci des droits supérieurs à ceux qu'il a sur les relations de la base servant à construire la vue. Les droits qui sont accordés automatiquement au créateur d'une vue sont égaux à l'ensemble des droits que l'usager possède sur chacune des relations composant la vue. Les droits acquis sur la vue ne sont transmissibles que s'ils l'étaient pour les relations de la base. Pour créer la vue, l'utilisateur doit avoir le droit de lecture (select) sur les relations servant à composer la vue. Il aura donc au minimum un droit de lecture sur la vue créée.