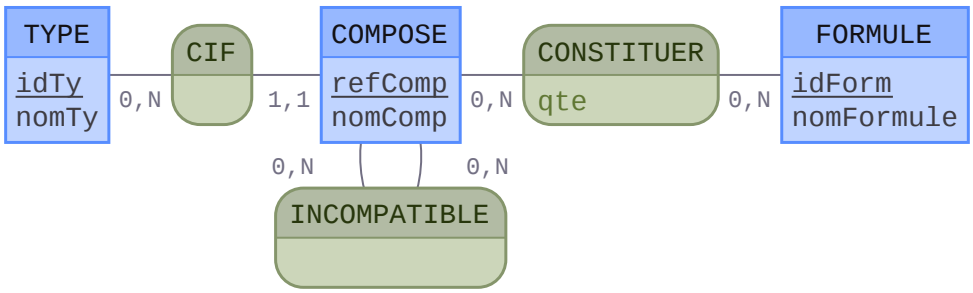
M2106: Bases de données avancées

Feuille de TP n°6

Les procédures et les fonctions stockées – Trigger

Une entreprise pharmaceutique possède la base de données suivante lui permettant de stocker les différentes formules qu'elle fabrique. Une formule est constituée de composés. Certains composés sont incompatibles et il est légalement interdit de les mettre ensemble dans une formule.



Le MLD correspondant est le suivant

Compose	(**refComp**, *nomComp*)
Constituer	(#***refComp***, #***idForm***, *qte*)
Formule	(**idForm**, *nomForm*)
Incompatible	(#***refComp.1***, #***refComp.2***)

Et le script de création des tables est le suivant

```
CREATE TABLE COMPOSE (
  refcomp varchar(4),
  nomcomp varchar(20),
  PRIMARY KEY (refcomp)
);

CREATE TABLE CONSTITUER (
  idform varchar(5),
  refcomp varchar(4),
  qte decimal(5,2),
  PRIMARY KEY (refcomp, idform)
);

CREATE TABLE FORMULE (
  idform varchar(5),
  nomform varchar(15),
  PRIMARY KEY (idform)
);

CREATE TABLE INCOMPATIBLE (
  refcomp_1 varchar(4),
  refcomp_2 varchar(4),
  PRIMARY KEY (refcomp_1, refcomp_2)
);

ALTER TABLE CONSTITUER ADD FOREIGN KEY (idform) REFERENCES FORMULE (idform);
ALTER TABLE CONSTITUER ADD FOREIGN KEY (refcomp) REFERENCES COMPOSE (refcomp);
ALTER TABLE INCOMPATIBLE ADD FOREIGN KEY (refcomp_1) REFERENCES COMPOSE (refcomp);
ALTER TABLE INCOMPATIBLE ADD FOREIGN KEY (refcomp_2) REFERENCES COMPOSE (refcomp);
```

Exercice 1 *Procédures et fonctions stockées (pour s'entraîner)*

Les deux première fonctions que nous allons écrire vont nous servir dans l'exercice suivant. Il s'agit d'énumérer les éléments d'une liste de chaînes de caractères séparées par des virgules. Par exemple, on considère la liste de couleurs suivante `'bleu,vert,rouge,jaune'`. On aimerait pouvoir énumérer chacune des couleurs de cette liste. Pour cela on va écrire deux fonctions :

1. `premier(liste varchar(255))` qui retourne le premier élément de la liste. Par exemple `premier('bleu,vert,rouge,jaune')` doit retourner `'bleu'`. Pour tester votre fonction vous pouvez taper dans le terminal MySQL

   ```
   select premier('bleu,vert,rouge,jaune');
   ```

2. `reste(liste varchar(255))` qui retourne la liste sans le premier élément. Par exemple `reste('bleu,vert,rouge,jaune')` doit retourner `'vert,rouge,jaune'`. S'il n'y a plus d'éléments dans la liste on retourne la chaîne vide `''`.
3. Pour tester vos fonctions vous allez créer une procédure `testerListe(liste varchar(255))` qui va
   — créer la table suivante

   ```
   create table TEST_TP6(elem VARCHAR(30));
   ```

   — puis insérer les éléments de la liste passée en paramètre de la procédure dans la table `TEST_TP6`.

Pour écrire ces fonctions vous aurez besoin des fonctions MySQL suivantes

— `locate(`*`ch1,ch2,n`*`)` qui recherche la position de la n$^{\text{ème}}$ occurrence de la chaîne *ch1* dans la chaîne *ch2*. Si cette occurrence n'existe pas la fonction retourne 0. Par exemple `locate(',','bleu,vert,rouge,jaune',1)` retourne 5.

— `substring(`*`ch,deb,long`*`)` qui extrait la partie de *ch* commençant à l'indice *deb* sur une longueur de *long* caractères. Par exemple `substring('bleu,vert,rouge,jaune',1,4)` retourne `'bleu'`.
Notez que `substring(`*`ch,pos`*`)` va extraire la fin de la chaîne à partir de l'indice *pos*.

Exercice 2 *Procédures et fonctions stockées (requêtes à une ligne)*

1. Écrire une fonction `nbFormules(nomCompose varchar(20))` qui retourne le nombre de formules qui utilisent le composé passé en paramètre.
2. Écrire une procedure `creerFormule(idF varchar(5), nomF varchar(15), listeComposes varchar(255))` qui va créer une nouvelle formule à partir des 3 paramètres, sachant que le troisième est la liste des noms des composés (séparés par des virgules) qui constituent la formule. Votre procédure devra donc insérer une nouvelle ligne dans la table `FORMULE` ainsi que les lignes nécessaires dans la table `CONSTITUER` (on mettra les quantités à 1.0). On considère que les composés existent déjà. Par exemple, on pourra créer la formule *Poison* à partir de la liste des composés *Mercure,Dioxyde de soufre,Sucre*

Exercice 3 *Les curseurs*

Afin de pouvoir éditer les étiquettes à coller sur les flacons contenant les différentes formules, l'entreprise a besoin d'une fonction `genereEtiquette(nomFormule varchar(15)) returns varchar(255)` qui retourne la liste des composants de la formule passée en paramètre sous la forme d'une chaîne de caractères. Par exemple pour l'étiquette de la formule *Comprimés*, la fonction doit retourner

```
Comprimés: Dioxyde de soufre(0.01) Dioxyde d'oxygène(1.20) Sucre(5.30)
```

où les valeurs entre parenthèses sont les quantités.

Exercice 4 *Les curseurs (suite)*

La table `INCOMPATIBLE` pour être cohérente doit être symétrique c'est-à-dire que si la ligne `('xxxx','yyyy')` est dans la table, il faut que la ligne `('yyyy','xxxx')` doit y être aussi. On vous demande d'écrire une procédure `completeIncompatible()` qui va rajouter toutes les lignes manquantes dans la table `INCOMPATIBLE`.

Exercice 5 *Les triggers : Historique*

Afin d'avoir des traces des mises à jour effectuées dans la base de données, l'entreprise a mis en place une table `HISTORIQUE` qui va enregistrer les différentes actions qui ont eu lieu sur les tables `FORMULE` et `COMPOSE`. Voici le script de création de la table historique et des triggers associés à la table `FORMULE` :

```
delimiter |
/* creation de la table HISTORIQUE */
drop table if exists HISTORIQUE|
create table HISTORIQUE( heure timestamp, action varchar(15), nomTable varchar(15),nouveau
    varchar(5), ancien varchar(5))|

/* trigger déclenché lors de l'insertion d'une nouvelle formule */
drop trigger if exists inserer_formule|
create trigger inserer_formule after insert on FORMULE for each row
begin
   insert into HISTORIQUE values (now(),'insertion', 'FORMULE', new.idForm,NULL);
end|

/* trigger déclenché lors de l'effacement d'une formule */
drop trigger if exists delete_formule|
create trigger delete_formule after delete on FORMULE for each row
begin
   insert into HISTORIQUE values (now(),'effacement', 'FORMULE', NULL, old.idForm);
end|

/* trigger déclenché lors de la mise à jour d'un formule */
drop trigger if exists update_formule|
create trigger update_formule after update on FORMULE for each row
begin
   insert into HISTORIQUE values (now(),'mise à jour', 'FORMULE', new.idForm, old.idForm);
end|
delimiter ;
```

1. Après avoir exécuté ce script sur votre base de données, observez le contenu de la table `HISTORIQUE` après les commandes suivantes :

   ```
   insert into FORMULE values ('XX001', 'XXXXXX');
   insert into FORMULE values ('YY001', 'YYYYYY');
   update FORMULE set idForm='XX002' where idForm='YY001';
   delete from FORMULE where idForm like 'XX%';
   ```

2. Créez les triggers qui permettent d'observer les modifications dans la table `COMPOSE`.

Exercice 6 *Les incompatibilités*

Écrire une trigger qui permet de vérifier lors de l'insertion dans le table `CONSTITUER` que le nouveau composé de la formule n'est pas incompatible avec les autres composés de cette même formule. Si il y a incompatibilité l'insertion doit être empêchée et un message d'erreur généré.

1. Ce trigger doit-il s'exécuter avant ou après l'insertion ?
2. Ce trigger doit-il s'exécuter une fois par commande insert ou une fois pour chaque ligne insérée ?

Exercices supplémentaires

Exercice 7 *Générateur d'identifiant de formules* (facultatif)

L'entreprise a pris comme convention pour les identifiants des formules de prendre les deux premières lettres du nom de la formule suivies du numéro d'ordre de la formule. Par exemple si les trois premières formules entrées dans la base de données sont *Sirop toux*, *Pastilles gorge* et *Pate dentifrice* on doit avoir les identifiants *SI001*, *PA002* et *PA003*.

1. Écrire une fonction `genereIdFormule(nomFormule varchar(15))` qui génère l'identifiant d'une formule en fonction du paramètre de la fonction et du contenu de la table `FORMULE`
2. Modifier votre procedure `creerFormule` de l'exercice précédent pour que l'identifiant automatiquement créer soit utilisé.

Pour écrire cette fonction vous aurez besoin des fonctions MySQL suivantes

— `convert(`*`valeur`*`,`*`type`*`)` qui convertit une valeur dans un type donné. Par exemple `convert('4',unsigned)` transforme la chaîne de caractère `'4'` en l'entier `4`. Symétriquement, `convert(4,char)` convertit l'entier `4` dans la chaîne de caractères `'4'`.

— `concat(`*`ch1`*`,`*`ch2`*`)` qui concatène les deux chaînes de caractères. Par exemple `concat('bon','jour')` donne `'bonjour'`

— `lpad(`*`ch`*`,`*`nbPad`*`,`*`padChar`* qui complète la chaîne de caractères *ch* pour qu'elle fasse une longueur *nbPad* en rajoutant à gauche autant de caractères *padChar* que nécessaire. Par exemple `lpad('4',3,0)` retourne la chaîne `'004'`.

Exercice 8 *Générateur d'identifiant de composés* (facultatif)

L'entreprise a pris comme convention pour les identifiants des composés de prendre les deux premières lettres du nom suivies d'un nombre pour distinguer les produits qui commencent par les même lettre. Par exemple, si on ajoute les composés suivants dans la base vide : Dioxyde de carbone, Mercure, Dioxyde de soufre ; on doit obtenir les identifiants suivants : Di01, Me01, Di02.
Écrire une fonction stockée `genereIdCompose(nomComp varchar(15))` qui génère l'identifiant du composé dont le nom est passé en paramètre en fonction du contenu de la table `COMPOSE`.