INSIA – SIGL 2
Bases de données
Cours 06 – PL-SQL
Procédures stockées - Triggers

Bertrand LIAUDET

SOMMAIRE

Première édition : Mars 2008

PL-SQL - PROCEDURES STOCKÉES - TRIGGERS

1. PL-SQL : les procédures stockées

http://dev.mysql.com/doc/refman/5.0/fr/stored-procedures.html

Présentation

Le SQL en général, et MySQL en particulier, permet d'écrire des procédures de programmation impérative classique (type Pascal, C, VB, PHP, etc.)

Ces procédures vont nous permettre, dans un premier temps, de développer un prototype de logiciel en réalisant toutes les fonctionnalités, avec une interface utilisateur restreinte.

Script d'exemple

```
-- script de définition d'une procédure
-- procédure « insertemp » : permet d'insérer un employé avec
son numéro,
-- son nom et son numéro de département

use empdept;
drop procedure if exists insertemp;
delimiter //

create procedure insertemp (v_ne integer, v_nom varchar(14),
v_nd integer)
begin
     insert into emp(ne, nom, nd) values (v_ne, v_nom, v_nd);
end ;
//
delimiter ;
```

Explications

- La procédure est créée avec l'instruction : « create procedure »
- Une liste de variable est passée en paramètre.
- Le corps de la procédure commence par « begin » et finit par « end ; »
- Dans le corps de la procédure on peut mettre des requêtes SQL et utiliser les paramètres de la procédure.
- Avant la création, il faut changer de délimiteur : delimiter //. Ceci vient du fait que la procédure utilise des « ; » comme délimiteur, et que le « ; » est le délimiteur standard du SQL.
- On termine la procédure par un délimiteur : //
- Après le //, il faut revenir au délimiteur standard : delimiter ;

Usage des procédures stockées : CALL

Le script précédent permet de créer la procédure « insertemp ».

Cette procédure s'utilise ainsi :

```
CALL insertemp (9500, « Durand », 10) ;
```

Le « call » permet d'appeler la procédure. On passe les paramètres à la procédure. L'instruction permet de créer un nouvel employé : Durant, n°9500 dans le département 10.

Gestion des procédures stockées

Afficher les procédures existantes :

```
Show procedure status ;
```

Afficher le code d'une procédure :

```
Show create procedure insertemp ;
```

2. PL-SQL : Eléments de programmation

Afficher du texte

```
/*   script de définition d'une procédure
     procédure « bonjour » : affiche bonjour,
     usage des commentaires en style C
*/

drop procedure if exists bonjour;
delimiter //
create procedure bonjour ( )
begin
     select 'bonjour';
end ;
//
delimiter ;
```

```
mysql> call bonjour();
+---------+
| bonjour |
+---------+
| bonjour |
+---------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Commentaires

```
/*   script de définition d'une procédure
     procédure « bonjour » : affiche bonjour,
     usage des commentaires en style C
*/

-- commentaires derrière deux tirets et un espace
```

Variables, type et affectation : DECLARE

```
drop procedure if exists testVariables;
delimiter //
create procedure testVariables ( )
begin
     declare my_int int;
     declare my_bigint bigint;
     declare my_num numeric(8,2);
     declare my_pi float default 3,1415926;
     declare my_text text;
     declare my_date date default '2008-02-01';
     declare my_varchar varchar(30) defaut 'bonjour';
     set my_int=20;
     set my_bigint = power(my_int,3);
     select my_varchar, my_int, my_big_int, my_pi, my_date,
my_num, my_text;
end ;
//
delimiter ;
```

```
mysql> call testVariables();
+------------+--------+-----------+---------+------------+--------
    +---------+
| my_varchar | my_int | my_bigint | my_pi   | my_date    | my_num |
    my_text |
+------------+--------+-----------+---------+------------+--------
    +---------+
| bonjour    |     20 |      8000 | 3.14159 | 2008-02-01 |   NULL |
    NULL    |
+------------+--------+-----------+---------+------------+--------
    +---------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.04 sec)
```

Tous les types en vrac :

Int, integer, bigint, décimal (nb1 chiffres max, nb2 chiffres après la virgule max), float (4 octets), double (8 octets), numeric(nb1, nb2), date, datetime, char(length), varchar(length), text, longtext, blob, longblob, enum, set.

```
drop procedure if exists testNum;
delimiter //
create procedure testNum ( )
begin
     declare my_dec decimal(8,2);
     declare my_num numeric(8,2);
     set my_dec=123456.789;
     set my_num = 123.456789e3;
     select my_dec, my_num;
end ;
//
delimiter ;
call testNum() ;
```

```
drop procedure if exists testEnumSet;
delimiter //
create procedure testEnumSet (inenum enum('oui','non','peut-
être'), inset set('oui','non','peut-être'))
begin
     declare position integer;
     set position=inenum;
     select inenum, position, inset;
end ;
//
delimiter ;
```

```
mysql> call testEnumSet('non', 'oui,peut-être');
+--------+----------+---------------+
| inenum | position | inset         |
+--------+----------+---------------+
| non    |        2 | oui,peut-être |
+--------+----------+---------------+
1 row in set (0.00 sec)
```

Opérateurs et fonctions accessibles

Les fonctions à utiliser : http://dev.mysql.com/doc/refman/5.0/fr/functions.html

- Opérateurs de comparaison : >, <, <=, >=, BETWEEN, NOT BETWEEN, IN, NOT IN, =, <>, !=, like, regexp (like étendu), isnull, is not null, <=>.
- Opérateurs mathématiques : +, -, *, /, DIV, %

- Opérateurs logiques : AND, OR, XOR
- Opérateurs de traitement de bit : | , &, <<, >>, ~
- Fonctions de contrôle : ifnull, if, case, etc.
- Fonctions de chaîne de caractères : substring, length, concat, lower, upper, etc.
- Fonctions numériques : abs, power, sqrt, ceiling, greatest, mod, rand, etc.
- Fonctions de dates et d'heures : current_date, current_time, to_days, from_days, date_sub, etc.
- Fonctions de recherche en texte intégral : match
- Fonctions de transtypage : cast et convert
- Autres fonctions

Paramètres en sortie : OUT et INOUT

Les paramètres des procédures sont en entrée par défaut : IN par défaut.

On peut spécifier un mode de passage en sortie : OUT ou INOUT s'il est en entrée-sortie.

La variable en sortie est récupérée comme variable globale de mysql.

```
drop procedure if exists my_sqrt;
delimiter //
create procedure my_sqrt(a int, OUT racine float)
begin
     set racine = sqrt(a);
end ;
//
delimiter ;
```

```
mysql> call my_sqrt(16, @res);
Query OK, 0 rows affected (0.00 sec)

mysql> select @res;
+------+
| @res |
+------+
| 4    |
+------+
1 row in set (0.00 sec)
```

Les variables de la calculette mysql

```
Mysql> select 'bonjour' into @x ;
Mysql> select @x;

Mysql> set @y='bonjour';
Mysql> select @y;

drop procedure if exists my_sqrt;
    delimiter //
    create procedure my_sqrt(a int, OUT racine float)
    begin
         set racine = sqrt(a);
    end ;
    //
    delimiter ;
```

Tests

```
drop procedure if exists soldes;
delimiter //
```

```
create procedure soldes(prix numeric(8,2), OUT prixSolde
numeric(8,2))
begin
     if (prix > 500) then
       set prixSolde=prix * 0.8 ;
     elseif (prix >100) then
       set prixSolde = prix * 0.9 ;
     else
       set prixSolde = prix ;
     end if ;
end ;
//
delimiter ;
```

```
mysql> call soldes(1000, @nouveauPrix);
Query OK, 0 rows affected (0.03 sec)

mysql> select @nouveauPrix;
+--------------+
| @nouveauPrix |
+--------------+
| 800.00       |
+--------------+
1 row in set (0.02 sec)
```

Syntaxe

IF *expression* THEN
instructions
[ELSEIF
expression
THEN
instructions …]
[ELSE
instructions]
END IF;

- ***Rappel de la sémantique du métalangage:***

Majuscule : les mots-clés

Entre crochets : ce qui est facultatif

Entre accolades : proposition de plusieurs choix possibles. Les choix sont séparés par des barres verticales.

3 petits points avant une accolade fermante : ce qui est entre accolades peut être répété.

En italitque : le nom d'une valeur, d'une variable, d'une expression, d'une instruction ou d'une série d'instructions.

Case

Syntaxe

CASE *expression* THEN
WHEN *valeurs* THEN
instructions

[WHEN *valeurs* THEN
instructions …]
[ELSE
instructions]
END CASE;

Boucles

Boucle sans fin et instruction "leave"

```
drop procedure if exists testBoucle;
delimiter //
create procedure testBoucle(n int, OUT somme int)
begin
     declare i int;
     set somme =0;
     set i = 1;
     maboucle : loop
       if ( i>n ) then
             leave maboucle;
       end if;
       set somme = somme + i;
       set i = i+1;
     end loop;
end ;
//
delimiter ;
```

```
mysql> call testBoucle(10, @res);
Query OK, 0 rows affected (0.00 sec)

mysql> select @res;
+------+
| @res |
+------+
| 55   |
+------+
1 row in set (0.00 sec)
```

- ***Syntaxe***

[*label :*] LOOP
instructions
END LOOP [*label*] ;

Boucle Repeat until

- ***Syntaxe***

[*label :*] REPEAT
instructions
UNTIL *expression* END REPEAT [*label*] ;

Boucle While

- ***Syntaxe***

[*label :*] WHILE *expression* DO

instructions

END WHILE [*label*] ;

Récupération de valeurs dans la BD : select into

```
drop procedure if exists testSelectInto;
delimiter //
create procedure testSelectInto(my_job varchar(9))
begin
     declare somSal float(7,2);
     select sum(sal) into somSal
     from emp
     where job = my_job;
     select somSal;
end ;
//
delimiter ;
```

```
mysql> call testSelectInto('CLERK');
+---------+
| somSal  |
+---------+
| 4150.00 |
+---------+
1 row in set (0.00 sec)
```

ou encore, avec le résultat en sortie

```
drop procedure if exists testSelectInto;
delimiter //
create procedure testSelectInto(my_job varchar(9), somSal
float(7,2))
begin
     select sum(sal) into somSal
     from emp
     where job = my_job;
end ;
//
delimiter ;
```

```
mysql> call testSelectInto('CLERK', @res);
Query OK, 0 rows affected (0.00 sec)
mysql> select @res;
+------+
| @res |
+------+
| 4150 |
+------+
1 row in set (0.01 sec)
```

Affichage de valeurs de la BD : select

```
drop procedure if exists testSelect;
delimiter //
create procedure testSelect(my_job varchar(9))
begin
     select my_job, avg(sal)
     from emp
     where job = my_job;
end ;
//
delimiter ;
```

```
mysql> call testSelect('CLERK');
+--------+-------------+
```

```
| my_job | avg(sal)    |
+--------+-------------+
| CLERK  | 1037.500000 |
+--------+-------------+
1 row in set (0.38 sec)
```

Les curseurs

http://dev.mysql.com/doc/refman/5.0/fr/cursors.html

Exemple avec une boucle loop

```
drop procedure if exists testCurseur;
    delimiter //
    create procedure testCurseur()
    begin
         declare i, vne int;
         declare vnom varchar(10);
         declare vjob varchar(9);

         declare vide int;
         declare curseur cursor for
           select ne, nom, job
           from emp
           where job='MANAGER';
         declare continue handler for not found set vide = 1 ;
         -- attention: si on met 0 ça boucle sans fin !!!

         set i=1;
         set vide=0;
         open curseur;
         maboucle: loop
           fetch curseur into vne, vnom, vjob;
           if vide=1 then
                 leave maboucle;
           end if;
           select i, vne, vnom, vjob;
           set i=i+1;
         end loop;
         close curseur;
    end ;
    //
    delimiter ;
```

- ***Résultats***

```
mysql> call testCurseur;
+------+------+-------+---------+
| i    | vne  | vnom  | vjob    |
+------+------+-------+---------+
|    1 | 7566 | JONES | MANAGER |
+------+------+-------+---------+
1 row in set (0.01 sec)

+------+------+-------+---------+
| i    | vne  | vnom  | vjob    |
+------+------+-------+---------+
|    2 | 7698 | BLAKE | MANAGER |
+------+------+-------+---------+
1 row in set (0.01 sec)

+------+------+-------+---------+
| i    | vne  | vnom  | vjob    |
+------+------+-------+---------+
|    3 | 7782 | CLARK | MANAGER |
+------+------+-------+---------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)
```

- ***Explicitations***

Déclaration d'un curseur : le curseur est un peu comme un pointeur qui se positionne sur la première ligne de la table associée au curseur, table définie par un select.

Déclaration d'une variable de type "continue handler for not found" : cette variable prendra la valeur 1 (vrai) quand on ne trouvera plus de ligne (tuple) dans la table associée au curseur ;

Initialisation de la variable « vide » à 0 : faux.

Open curseur : le curseur est positionné sur le premier élément de la table. Si la table est vide, le curseur est donc à la fin.

fetch : permet de récupérer la valeur de chaque attribut de la ligne en cours de la table correspondant au curseur. Le fetch positionne le curseur sur la ligne suivante pour le fetch suivant. Si on fait un fetch alors que le curseur est positionné sur la fin de la table, alors le « continue handler for not found » prend la valeur prévue dans la déclaration : ici vide passe à 1 (vrai).

Close curseur : ça libère l'accès aux données pour d'autres utilisateurs.

Blocs imbriqués

On peut déclarer des blocs d'instructions à tout moment dans une procédure.

Quand on déclare un bloc d'instruction, on peut y associer de nouvelles déclarations de variables.

- ***Syntaxe***

```
[ label : ] BEGIN
        [ déclaration de variable  . . .] ;
        instructions
END [ label ] ;
```

Exemple avec une boucle loop

```
drop procedure if exists testCurseur;
```

3. PL-SQL : Les fonctions stockées

http://dev.mysql.com/doc/refman/5.0/fr/stored-procedures.html

Présentation

Les fonctions n'ont qu'un paramètre en sortie qui est renvoyé par le return.

Donc, tous les paramètres de l'en-tête sont en entrée : on ne le précise pas.

```
drop function if exists testFonction;
delimiter //
create function testFonction(my_job varchar(9))
     returns float(7,2)
begin
     declare somSal float(7,2);
     select sum(sal) into somSal
     from emp
     where job = my_job;
     return (somSal);
end ;
//
delimiter ;
```

Usage des fonctions stockées

Les fonctions, comme toute fonction, peuvent s'utiliser dans n'importe quelle expression à la place d'une variable ou d'une valeur du type renvoyé par la fonction.

```
mysql> select testFonction('CLERK');
+-----------------------+
| testFonction('CLERK') |
+-----------------------+
|               4150.00 |
+-----------------------+
1 row in set (0.66 sec)
```

Gestion des fonctions stockées

Afficher les fonctions existantes :

```
Show function status ;
```

Afficher le code d'une fonction :

```
Show create function testFonction ;
```

4. PL-SQL : Les triggers (déclencheurs)

http://dev.mysql.com/doc/refman/5.0/fr/triggers.html

Présentation

Un trigger est une procédure stockée qui est déclenchée automatiquement avant (BEFORE) ou après (AFTER) l'insertion (INSERT), la modification (UPDATE) ou la suppression (DELETE) d'un tuple (une action du DML).

Trigger BEFORE et trigger AFTER

Un trigger BEFORE est un trigger qui se déclenche avant l'action du DML (INSERT, UPDATE ou DELETE).

Un trigger BEFORE sert soit à vérifier l'intégrité de la BD du fait de la modification qu'on souhaite apporter. Toutefois, la vérification se fait avant d'avoir enregistré les nouvelles données. Le trigger peut alors soit interdire la modification, soit la modifier, soit la laisser s'exécuter mais envoyer un message d'avertissement.

Un trigger AFTER sert à mettre à jour la BD du fait de la modification qu'on vient d'y apporter. La mise à jour se fait donc après avoir enregistré les nouvelles données. Le trigger AFTER ne peut pas modifier de données dans sa propre table (il ne peut donc pas remplacer le trigger BEFORE).

Unicité des triggers

On ne peut avoir qu'un et un seul trigger BEFORE par action de DML pour une table donnée.

On ne peut avoir qu'un et un seul trigger AFTER par action de DML pour une table donnée.

Gestion des triggers

Afficher les triggers existants :

```
Show triggers ;
```

Afficher le code d'un trigger:

```
Show triggers ;
```

Exemple de TRIGGER AFTER

Dans la BD biblio, on ajoute le champ « dispo » dans la table LIVRES : c'est un champ calculé qui dit si le livre est disponible ou pas.

```
drop trigger if exists afterinsert_emprunter;
delimiter //
create trigger afterinsert_emprunter
after insert on emprunter
for each row
begin
update livres set dispo=0 where nl=new.nl;
-- select new.nl into @vnl; si on veut verifier la valeur de new.nl
end;
//
```

```
delimiter ;
```

Syntaxe

CREATE [DEFINER={*user* | CURRENT_USER}] TRIGGER *nomTrigger*
{ BEFORE | AFTER } { UPDATE | INSERT | DELETE } ON *tableName*
FOR EACH ROW
InstructionsDuTrigger

Rappel de la sémantique du métalangage:

Majuscule : les mots-clés

Entre crochets : ce qui est facultatif

Entre accolades : proposition de plusieurs choix possibles. Les choix sont séparés par des barres verticales.

En italique : le nom d'une variable ou d'une série d'instructions.

Précisions

[DEFINER={*user* | CURRENT_USER}] : permet de limiter l'usage du trigger à un utilisateur.

nomTrigger : c'est le nom donné à la procedure trigger.

{ BEFORE | AFTER } : choix du déclenchement avant ou après l'opération du DML.

{ UPDATE | INSERT | DELETE }: choix de l'opération du DML.

tableName : nom de la table à laquelle est associé le trigger.

donnée à la procedure trigger.

FOR EACH ROW : le trigger s'applique à tous les tuples affectés par les traitements. MySQL ne permet que cette option.

Exemple de TRIGGER BEFORE

Dans la BD employés, on ajoute le champ « saltot » dans la table EMPLOYES : c'est un champ calculé qui calcule le salaire total (sal + comm).

```
drop trigger if exists beforeinsert_emp;
    delimiter //
    create trigger beforeinsert_emp
         before insert on emp
         for each row
    begin
         -- sal est un attribut not NULL
         set new.saltot=new.sal+ifnull(new.comm,0);
    end ;
    //
    delimiter ;
```

new

new est un mot-clé : il précise sur quel tuple on travaille : celui qui va être inséré.

Usage des triggers

Les triggers sont déclenchés automatiquement quand l'événement DML déclencheur arrive :

```
mysql> insert into emp (NE, nom, sal, ND) values (9999, 'TOTO', 1200, 10);
Query OK, 1 row affected (0.17 sec)

mysql> select * from emp;
+------+--------+-----------+------------+---------+---------+----+-------+---------+
| NE   | NOM    | JOB       | DATEMB     | SAL     | COMM    | ND |NEchef | saltot |
+------+--------+-----------+------------+---------+---------+----+-------+---------+
| 7369 | SMITH  | CLERK     | 1980-12-17 |  800.00 |    NULL | 20 |  7902 |  800.00|
| 7499 | ALLEN  | SALESMAN  | 1981-02-20 | 1600.00 |  300.00 | 30 |  7698 | 1900.00|
| 7521 | WARD   | SALESMAN  | 1981-02-22 | 1250.00 |  500.00 | 30 |  7698 | 1750.00|
| 7566 | JONES  | MANAGER   | 1981-04-02 | 2975.00 |    NULL | 20 |  7839 | 2975.00|
| 7654 | MARTIN | SALESMAN  | 1981-09-28 | 1250.00 | 1400.00 | 30 |  7698 | 2650.00|
| 7698 | BLAKE  | MANAGER   | 1981-05-01 | 2850.00 |    NULL | 30 |  7839 | 2850.00|
| 7782 | CLARK  | MANAGER   | 1981-06-09 | 2450.00 |    NULL | 10 |  7839 | 2450.00|
| 7788 | SCOTT  | ANALYST   | 1982-12-09 | 3000.00 |    NULL | 20 |  7566 | 3000.00|
| 7839 | KING   | PRESIDENT | 1981-11-17 | 5000.00 |    NULL | 10 |  NULL | 5000.00|
| 7844 | TURNER | SALESMAN  | 1981-09-08 | 1500.00 |    0.00 | 30 |  7698 | 1500.00|
| 7876 | ADAMS  | CLERK     | 1983-01-12 | 1100.00 |    NULL | 20 |  7788 | 1100.00|
| 7900 | JAMES  | CLERK     | 1981-12-03 |  950.00 |    NULL | 30 |  7698 |  950.00|
| 7902 | FORD   | ANALYST   | 1981-12-03 | 3000.00 |    NULL | 20 |  7566 | 3000.00|
| 7934 | MILLER | CLERK     | 1982-01-23 | 1300.00 |    NULL | 10 |  7782 | 1300.00|
| 9999 | TOTO   | NULL      | NULL       | 1200.00 |    NULL | 10 |  7839 | 1200.00|
+------+--------+-----------+------------+---------+---------+----+-------+---------+
16 rows in set (0.01 sec)
```

Débogage

On ne peut pas afficher des commentaires dans un trigger via un select.

Par contre, on peut utiliser des variables globales et faire un select dans une variable globale.

Trigger et vérification des contraintes d'intégrité

Les triggers doivent aussi permettre de bloquer une action du DML si cette action n'est pas conforme aux contraintes d'intégrité définies la conception.

```
drop trigger if exists testTriggerErreur;
    delimiter //
    create trigger testTriggerErreur
         before insert on emp
         for each row
    begin
         declare  mesErr int;
         select 'parici' into @parici;
         if new.sal  < 0 then
            select 'le salaire ne peut pas être négatif' into mesErr
            from emp;
         end if;
    end ;
    //
    delimiter ;
```

```
mysql> insert into emp (nom, sal, ND) values ('DURAND', -100, 10);

ERROR 1366 (HY000): Incorrect integer value: 'le salaire ne peut
    pas être négatif' for column 'mesErr' at row 1

mysql> insert into emp (nom, sal, ND) values ('DURAND', 100, 10);

Query OK, 1 row affected (0.13 sec)
```

On pourrait faire la même chose sur un update.

Remarque : l'instruction SIGNAL

A la place de la solution précédente, qui n'est pas très élégante, le SQL-ANSI définit l'instruction SIGNAL qui permet de gérer l'interruption et les messages.

Mais cette solution n'est pas opérationnelle sous MySQL.

Exemple :

```
drop trigger if exists testTriggerErreur;
    delimiter //
    create trigger testTriggerErreur
         before insert on emp
         for each row
    begin
         declare  mesErr int;
         if new.sal  < 0 then
           -- Attention ! Pas implémenté en MySQL 5.0
           SIGNAL SQLSTATE '80000' SET MESSAGE_TEXT= 'le salaire ne
           peut pas être négatif' ;
         end if;
    end ;
    //
    delimiter ;
```

TP PROCEDURES STOCKEES ET TRIGGERS

1. La bibliothèque – procédures stockées

1) Créer la BD « biblio » à partir du script fourni.
2) Créer la procédure stockée qui permette de rechercher les adhérents par leur nom et éventuellement par prénom.
3) Créer la procédure stockée qui permette de créer un nouvel adhérent.
4) Créer la procédure qui permette de supprimer un adhérent
5) Créer la procédure stockée qui permette de modifier les caractéristiques d'un adhérent déjà enregistré.
6) Créer la ou les procédures stockées qui permette d'ajouter un nouveau livre (traiter tous les cas possibles).
7) Créer la procédure qui permette de supprimer une œuvre. Supprimer l'œuvre : « Narcisse et Goldmund ».
8) Créer la procédure stockée qui permette d'enregistrer un emprunt. Mettez la valeur de durée max comme valeur par défaut. Modifier la procédure.
9) Créer la procédure stockée qui permette d'enregistrer un retour.

2. La bibliothèque - triggers

1) Créer la BD « biblio » à partir du script fourni.
2) On souhaite ajouter l'attribut « emprunté » qui est un booléen et précise si un livre est actuellement emprunté ou pas. Passez la requête qui permet d'ajouter cet attribut (ALTER TABLE).
3) Mettre à jour les valeurs de cet attribut (UPDATE).
4) Ecrire les triggers qui permettra de gérer cet attribut calculé.
5) On souhaite que la date d'emprunt soit désormais un attribut automatique qui vaut automatiquement la date du jour de la création du tuple correspondant dans la BD. Comment faire ça ? Ecrire le code correspondant.
6) On souhaite interdire l'emprunt d'un livre déjà empruté. Ecrire le trigger.
7) On souhaite interdire l'emprunt de plus de 3 livres par le même adhérent. Ecrire le trigger.
8) On souhaite que la date de retour prenne automatiquement la valeur de la date du jour du rendu du livre. On souhaite interdire un rendu le jour même de l'emprunt. On souhaite interdire toute modification des emprunts une fois le livre rendu. Ecrire le trigger.
9) Après analyse, on décide de fusionner les tables LIVRES et OEUVRES en une seule table : écrire le nouveau script de création de la BD qui intégrera cette modification.
10) Ecrire le ou les triggers qui vérifient la cohérence des données suite à cette modification.

3. Les employés et les départements

1) Créer la BD « empdept » à partir du script fourni.
2) On souhaite ajouter l'attribut « Salaire total » qui donne la somme du salaire et de la commission : passez la requête qui permet d'ajouter cet attribut (ALTER TABLE).
3) Mettre à jour les valeurs de cet attribut (UPDATE).
4) Ecrire le trigger qui permettra de gérer cet attribut calculé.
5) Après analyse, on décide de fusionner les tables EMP et DEPT en une seule table : écrire le nouveau script de création de la BD qui intégrera aussi l'attribut calculé « salaire total ».
6) Ecrire le ou les triggers qui vérifient la cohérence des données suite à cette modification.

4. Système d'enchères électroniques

On souhaite développer un système d'enchères électroniques. Les premières fonctionnalités décrites sont les suivantes :

Tout utilisateur du système (vendeur ou acheteur) doit préalablement s'inscrire en fournissant une adresse email qui lui servira d'identificateur unique pour ses futures interventions, un mot de passe choisi par lui qui lui servira d'authentification pour ses futures interventions et des renseignements d'état-civil : nom, prénom, adresse.

Les utilisateurs peuvent mettre en vente des objets. Pour cela, ils doivent fournir leur identificateur, leur mot de passe, la catégorie de l'objet pour une recherche par thème, un intitulé court de l'objet, une description détaillée de l'objet avec ses défauts et ses qualités, une mise à prix en euros, la date de début de l'enchère, la durée de l'annonce en jours, une ou plusieurs photos, s'il le souhaite. Le système fournit en retour un numéro de lot unique pour l'objet à vendre.

Un utilisateur ayant proposé un objet à la vente peut réviser la mise à prix à la baisse tant qu'il n'y a aucunes enchères sur l'objet.

Les utilisateurs peuvent effectuer des recherches parmi les objets mis en vente selon les critères suivants : catégorie de l'objet, mots de la description de l'objet, fourchette de prix.

Un utilisateur peut enchérir sur un objet. Pour être valable, l'enchère doit être supérieure de 1 à la plus haute enchère déjà effectuée ou à la mise à prix si c'est la première enchère ; l'annonce ne doit pas être expirée ; l'utilisateur ne doit pas être le vendeur de l'objet. Pour suivre l'évolution de l'enchère, le système garde aussi la date de chaque enchère.

Quand l'échéance de fin d'enchère tombe, le système informe le vendeur dans tous les cas, et l'acheteur s'il y a lieu, du résultat de l'enchère.

Si un vendeur veut remettre en vente un objet non vendu, il doit créer un nouvel objet dans le système avec ses caractéristiques propres.

Par ailleurs, les utilisateurs peuvent aussi sélectionner certains objets dont ils veulent suivre l'enchère. Le système enverra un mail d'alerte à l'utilisateur 24 heures avant la fin de l'enchère. L'utilisateur pourra supprimer les éléments de sa sélection.

1) Créer la BD correspondant à ce cahier des charges.
2) Ajouter les contraintes d'intégrité.

3) Quels attributs calculés peut-on envisager ? Coder l'ajout de ces attributs.
4) Quelle fusion de tables peut-on envisager ? Coder la fusion des tables.