INSIA
Bases de données
PL-SQL
Procédures et fonctions stockées sous MySQL

Bertrand LIAUDET

SOMMAIRE

Première édition : Mars 2008
Deuxième édition : Octobre 2009 – Mise à jour septembre 2011

PL-SQL - PROCEDURES ET FONCTIONS STOCKEES

1. PL-SQL : les procédures stockées

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

Présentation

Les SGBD-R en général, et MySQL en particulier, permettent 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)
comment 'permet d''insérer un employé avec ses numéro, nom et
n° de dept'
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.
- Des commentaires sont ajoutés avec le « comment »
- 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 ;
```

Supprimer une procédure :

```
Drop procedure insertemp ;
```

Créer une procédure :

```
Create procedure insertemp...
;
```

http://dev.mysql.com/doc/refman/5.0/fr/create-procedure.html

Gestion des erreurs

Afficher les erreurs-warnings :

```
Show warnings ;
```

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';
     select 'tout le monde';
end ;
//
delimiter ;
```

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

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

Afficher une table

```
...
begin
     select * from emp;
end ;
//
delimiter ;
```

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 locales, déclaration, 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) default 'bonjour';
     set my_int=20;
     set my_bigint = power(my_int,3);
     select my_varchar, my_int, my_bigint, 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 (0.00 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.

- ***Exemple 1 : decimal et numérique***

```
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() ;
```

```
mysql>  call testNum() ;
+-----------+-----------+
| my_dec    | my_num    |
+-----------+-----------+
| 123456.79 | 123456.79 |
+-----------+-----------+
1 row in set (0.00 sec)
```

➢ ***Exemple 2 : enum et set***

```
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; //un enum est une val. et une position
     select inenum, position, inset;
end ;
//
delimiter ;
call testEnumSet('non', 'oui,peut-être');
```

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

Variables globales

```
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 ;
```

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 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 peut être 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)
```

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.

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

Case

Syntaxe

```
CASE expression
WHEN valeurs THEN
    instructions
[WHEN valeurs THEN
    instructions ]
[ELSE
    instructions ]
```

Boucles

Boucle While

- ***Syntaxe***

[*label :*] WHILE *expression* DO
instructions
END WHILE [*label*] ;

- ***Exemple***

```
drop procedure if exists testWhile;
delimiter //
create procedure testWhile(n int, OUT somme int)
comment 'testWhile'
begin
        declare i int;
        set somme =0;
        set i = 1;
        while i <=n do
          set somme = somme + i;
          set i = i+1;
        end while;
end;
//
delimiter ;
```

Boucle Repeat until

- ***Syntaxe***

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

Boucle sans fin

- ***Syntaxe***

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

Sortie de boucle: leave

- ***Exemple : boucle avec compteur : 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 maboucle;
     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***

[*monLabel :*] *DEBUT_DE_BOUCLE*
    LEAVE *monLabel ;*
FIN_DE_BOUCLE [*monLabel*] ;

L'instruction « leave » permet de quitter n'importe quel bloc d'instruction précédé par le nom d'un label. C'est un « goto » structuré.

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*] ;

Sortie de bloc : leave

- ***Syntaxe***

[*monLabel :*] *DEBUT_DE_BLOC*
    LEAVE *monLabel ;*
FIN_DE_BLOC [*monLabel*] ;

L'instruction « leave » permet de quitter n'importe quel bloc d'instruction précédé par le nom d'un label. C'est un « goto » structuré.

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)
```

Utilisation de commandes du DDL et du DML

On peut passer toutes les commandes du DDL, du DML et du DCL (grant, revoke, commit, rollback) à une procédure.

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

create procedure testDDML ()
begin
     drop table if exists test;
     create table test (num integer);
     insert into test values(1), (2), (3);
     select * from test;
end ;
//
delimiter ;
```

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';
    -- le continue handler permettra de contrôler les fetch
    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)
```

- ***Explications***

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.

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)
     deterministic
begin
     declare somSal float(7,2);
     select sum(sal) into somSal
     from emp
     where job = my_job;
     return (somSal);
end ;
//
delimiter ;
```

Remarque

Le mot clé « deterministic » est obligatoire bien qu'il ne serve à rien !

http://dev.mysql.com/doc/refman/5.0/fr/create-procedure.html

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 ;
```

TP PROCEDURES ET FONCTIONS STOCKEES

0. MySQL Query Browser

http://dev.mysql.com/downloads/gui-tools/5.0.html

Pour écrire les procédures stockées et triggers, on peut utiliser le MySQL Query Browser.

1. La bibliothèque – procédures et fonctions stockées

1) Créer la BD « biblio » à partir du script fourni.
2) Ecrire une fonction qui calcule, pour un adhérent donné, le nombre de jours restant avant d'être en retard. Si l'adhérent n'a pas d'emprunts en cours, on renvoie NULL. Si l'adhérent est en retard, on renvoie un résultat négatif. On prendra en compte la possibilité d'avoir des emprunts avec des dureeMax différentes et des emprunts en cours avec des dates d'emprunt différentes. Dans ce cas, on renverra le nombre de jours restant le plus petit et le nombre de jours de retard le plus grand.
3) Utiliser cette fonction pour afficher la situation de tous les adhérents.
4) Ecrire une procédure qui permette de lister les emprunts d'un adhérent identifié par son numéro.
5) Ecrire une procédure qui affiche les exemplaires disponibles d'un titre.
6) Ecrire une procédure qui affiche les titres d'un auteur et le nombre d'exemplaires disponibles par titre.
7) Ecrire une procédure qui permette d'enregistrer un emprunt.
8) Modifier la table des emprunts : mettez la valeur par défaut de la durée max à 14.
9) Ecrire une nouvelle procédure qui enregistre un emprunt et gère tous les cas d'erreur (le livre n'existe pas, l'adhérent n'existe pas, le livre est déjà emprunté, l'adhérent emprunte déjà 3 livres, etc.). La procédure impose la date du jour comme date d'emprunt. La procédure renverra un numéro de code pour chaque erreur.
10) Rajouter le trigger qui permet de gérer l'attribut « emprunté », booléen permettant de savoir si un livre est emprunté ou pas (cf. TP précédent).
11) Vérifier que ce trigger fonctionne avec la procédure stockée de l'exercice précédent.
12) Créer la procédure stockée qui permette d'enregistrer un retour en gérant tous les cas d'erreur (le livre n'existe pas, l'adhérent n'existe pas, le livre n'est pas emprunté, etc.) et en imposant la date du jour comme date de retour. La procédure renverra un numéro de code pour chaque erreur et le nombre de jour de retard s'il y a lieu.

2. Les chantiers – procédure stockée

On souhaite enregistrer des bilans sur l'utilisation des voitures. On va conserver, pour chaque voiture, le nombre de visites, le nombre de passager et le nombre de kilomètres effectués par mois.

Créer une table qui permet d'enregistrer ces informations.

Cette table est mise à jour une fois par an. Ecrire une procédure qui permet de remplir cette table à partir des informations qui sont dans la base. La procédure permettra aussi de mettre à jour l'attribut « kilométrage » des véhicules (on lui ajoute les kilomètres parcourus à chaque visite). Enfin la procédure supprimera toutes les visites de l'année.

On utilisera préférentiellement un curseur.

Peut-on se passer d'un curseur ?

3. BD Ecoling - fonction stockée

Charger la BD Ecoling.

Dans la BD Ecoling, écrire une fonction qui permet d'afficher, pour un examen donné, la moyenne des notes, la meilleure note et le ou les noms des élèves, la plus basse note et le ou les noms des élèves (group_concat), pour chacune des épreuves.

4. Programmation classique – procédures stockées

Ecrire un programme qui permet de résoudre une équation du second degré.

On écrira d'abord une procédure qui résout une équation du premier degré avec l'entête suivante : a (in), b(in), x(out), nbsol(out) ; avec nbsol = -1 dans le cas d'une infinité de solutions.

La procédure équa2 fera appel à la procédure équa1

On écrira une procédure qu sert de programme principal et qui gère l'interface utilisateur.

On testera le programme avec tous les cas possibles : 2 solutions, 1 solution double, 0 solution type équa2, 1 solution simple, 0 solution type équa1, une infinité de solutions.