Bases de données
PL-SQL
MySQL – Procédures et fonctions stockées

Bertrand LIAUDET

SOMMAIRE

Edition sept 2019

PL-SQL - PROCEDURES ET FONCTIONS STOCKEES

1. PL-SQL : les procédures stockées

https://dev.mysql.com/doc/refman/8.0/en/faqs-stored-procs.html

https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-data-definition.html

Présentation

Les SGBD-R en général, et MySQL en particulier, permettent d' **écrire des procédures et des fonctions de programmation impérative classique** (type Pascal, C, VB, PHP, etc.) qui seront **enregistrées dans la base de données**.

Usages

Écriture de fonctions

On peut écrire des **fonctions de calcul plus moins complexes** qu'on pourra ensuite utiliser dans nos SELECT et commandes DML.

Orienté Objet

Les procédures stockées permettent de fournir un **jeu de méthodes associées à une table**, comme pour une classe en programmation objet. Ca peut être au moins les INSERT, UPDATE, DELETE, mais aussi n'importe quel SELECT. On peut ensuite ne donner l'accès qu'à ces procédures stockées aux développeurs.

Ces procédures (et fonctions) permettent de développer des méthodes d'utilisation de la BD, méthode qui cachent un accès direct aux données. Elles jouent alors le rôle des méthodes dans les classes de la programmation objet.

Les procédures vont encapsuler les attributs et permettre certains usages particuliers.

Archivage et synthèse

Les procédures stockées peuvent aussi servir pour gérer des **tâches d'archivage** récurrentes qui permettent de vider la base.

Elles permettent aussi de faire des synthèses (annuelles) par exemple qu'on enregistre dans de nouvelles tables créées par les procédures.

Orienté Admin

Les procédures stockées permettent de **faire de l'administration** en utilisant par exemple les données du dictionnaire des données.

Méthode de codage

Principes généraux de codage

Dans une procédure PL-SQL, on peut utiliser les techniques de programmation procédurale classique : **variable, test, boucle et fonction**.

On peut aussi intégrer les **commandes SQL**.

On trouvera les notions de « **curseur** » et de « **fetch** » qui permettent de **parcourir les lignes d'un SELECT**.

SQL versus Curseur

Autant que possible, mieux vaut **utiliser les possibilités du SQL** (select, attributs calculés, group by, jointure, etc.) **plutôt que de passer par des curseurs**.

Ca permet une **meilleure optimisation** des calculs et un code plus court et plus lisible.

Script d'exemple de type « orienté objet » (exemple 1)

```
-- script de définition d'une procédure
-- procédure « insertemp » : permet d'insérer un employé avec :
-- son numéro, son nom et son numéro de département
-- les autres attributs doivent ne pas être obligatoires !

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
    /* on insert dans la BD
    c'est juste un exemple de code
    */
    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 ».** Ces commentaires sont enregistrés dans la BD. On s'en sert pour décrire le fonctionnement général de la procédure ou de la fonction.
- **/* */** : ces **commentaires ne seront pas enregistrés dans la BD.**
- 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

https://dev.mysql.com/doc/refman/8.0/en/call.html

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 erreurs

Afficher les erreurs-warnings :

```
Show warnings ;
```

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...
;
```

Procédures stockées dans le dictionnaire des données : information_schema

Afficher les procédures existantes :

```
Select  routine_name
from    information_schema.routines ;
```

Afficher le code d'une procédure :

```
Select  routine_definition
from    information_schema.routines
where   routine_name = 'insertemp';
```

Afficher le comment d'une procédure :

```
Select  routine_comment
from    information_schema.routines
where   routine_name = 'insertemp';
```

Afficher les paramètres d'une procédure :

```
select  *
from    information_schema.parameters
where   specific_name='insertemp'; \G
```

Le \G permet d'avoir un affichage « page » avec une ligne par attribut : c'est plus lisible.

2. PL-SQL : Eléments de programmation

Afficher du texte (exemple 2)

```
/* script de définition d'une procédure stockée
   procédure « bonjour » :
      affiche "bonjour" et "tout le monde"
   usage des commentaires en style C
*/
drop procedure if exists bonjour;
delimiter //
create procedure bonjour ( )
begin
   /* commentaires au choix */
   select "bonjour";
   select "tout le monde";
end ;
//
delimiter ;
```

Les commentaires classiques /* */ passent dans l'interpréteur.

```
mysql> call bonjour();
+---------+
| bonjour |
+---------+
| bonjour |
+---------+
1 row in set (0,00 sec)

+---------------+
| tout le monde |
+---------------+
| tout le monde |
+---------------+
1 row in set (0,00 sec)
```

Afficher une table

Principe

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

Commentaires

https://dev.mysql.com/doc/refman/8.0/en/comments.html

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

-- commentaires derrière deux tirets et un espace

/*   */ commentaire dans la ligne

# commentaire de fin de ligne
```

Commentaires généraux sur la procédure ou la fonction

Entre le create et le begin, on met une ligne commençant par « comment ».

L'information est enregistrée dans la BD

```
comment 'commentaire conservée dans la BD'
```

Variables locales : déclaration, type et affectation

DECLARE: déclaration

```
DECLARE my_int int;
```

SET : affectation

```
SET my_int=20;
```

SELECT … INTO : affectation

```
SELECT count(*) from emp INTO my_int;
```

ou

```
SELECT count(*) INTO my_int from emp;
```

et

```
SELECT ne, nom from emp where ne=7369 INTO my_ne, my_nom;
```

ou

```
SELECT ne, nom INTO my_ne, my_nom from emp where ne=7369;
```

Exemple (exemple 3)

```
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)
```

Variables globales : @

Déclaration et affectation : @ et SET

```
set @gbint=99;
set my_int=@gbint;
```

Déclaration et affectation : @ et SELECT ... INTO

```
Select nom into @gbchar from emp limit 1;
```

Affichage d'une variable globale dans un programme

```
Select @gbchar;
```

Affichage d'une variable globale dans le SGBD

```
mysql > Select @gbchar;
```

Exemple

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

set @y='hello';
select @y;

select count(*) from emp into @z ;
select @z;

Select ne, nom into @my_ne, @my_nom from emp where ne=7369;
Select @my_ne, @my_nom;
```

Paramètres en entrée : IN

Exemple

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

Exemple 1

```
drop procedure if exists testSelect;
delimiter //
create procedure testSelect(IN 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)
```

Exemple 2

```
    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)
```

Paramètres en sortie : OUT et INOUT (exemple 4)

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(IN 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)
```

Utilisation de commandes du DDL, DML et DCL

Utilisation « statique »

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

Utilisation « dynamique » (exemple 5)

https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-prepared-statements.html

On peut passer des variables aux commandes SELECT, DDL, DML, DCL qu'on exécute dans une procédure stockées.

Ca passe par une syntaxe particulière consistant à **fabriquer la requête comme une chaine** en y insérant des variables.

Ensuite on « **prépare** » et on « **exécute** » la requête (on retrouve ça en PHP, par exemple).

A la fin on libère les tables utilisées (**deallocate**).

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

create procedure testEXECUTE (my_attribut varchar(20))
begin
     SET @requete = CONCAT('SELECT ',my_attribut,' from emp; ');

     -- pour vérifier
     Select @requete;

     -- méthode d'exécution dynamique :
     -- PREPARE, EXECUTE, DEALLOCATE
     PREPARE req FROM @requete;
     EXECUTE req;
     DEALLOCATE PREPARE req;
end ;
//
delimiter ;

call testEXECUTE('ne') ;
call testEXECUTE('nom') ;
call testEXECUTE('ne, nom') ;
```

- ***Remarques :***

« **requete** » est une variable globale.

« **req** » n'est pas déclarée.

On réabordera cette usage avec la gestion du dictionnaire.

Tous les types

https://dev.mysql.com/doc/refman/8.0/en/data-types.html

Types numériques

INT, integer, bigint

DECIMAL (nb1 chiffres max, nb2 chiffres après la virgule max) : virgule fixe, pour les Mathématiques de précision, équivalent à NUMERIC, DEC et FIXED,

FLOAT (4 octets), REAL, DOUBLE PRECISON (8 octets) : virgule flottante.

Les types date et heure

DATETIME, TIMESTAMP (stockage automatique sur insert et update)

YEAR, DATE, TIME

Les types chaînes

CHAR(length) et **VARCHAR**(length) : des caractères.

BINARY(length) and VARBINARY(length) : des octets (equivalent char, varchar).

BLOB et **TEXT** (longblob, longtext): Binary Long Object (des octets), TEXT (des caractères) : en gigas.

ENUM : 1 valeur choisie parmi une liste de valeurs autorisées.

SET : plusieurs valeurs choisies parmi une liste de valeurs autorisées.

Capacité des colonnes

http://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html

Exemples

- ***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 (in_enum enum('oui','non','peut-
être'), in_set set('oui','non','peut-être'))
begin
     declare position integer;

     set position=in_enum; //un enum est une val et une position
     select in_enum, position, in_set;
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)
```

Opérateurs et fonctions accessibles

Les fonctions à utiliser : http://dev.mysql.com/doc/refman/5.6/en/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

Tests – IF – ELSE – ELSEIF

https://dev.mysql.com/doc/refman/8.0/en/if.html

```
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       |
+--------------+
```

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 WHEN

https://dev.mysql.com/doc/refman/8.0/en/case.html

Syntaxe

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

Boucles

Boucle While

https://dev.mysql.com/doc/refman/8.0/en/while.html

- ***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

https://dev.mysql.com/doc/refman/8.0/en/repeat.html

- ***Syntaxe***

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

Boucle sans fin

https://dev.mysql.com/doc/refman/8.0/en/loop.html

- ***Syntaxe***

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

Sortie de boucle: leave

https://dev.mysql.com/doc/refman/8.0/en/leave.html

- ***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

De même qu'on a des « leave » pour les blocs de boucle, on peut avoir des « leave » pour tous les blocs.

- ***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.

Les curseurs

https://dev.mysql.com/doc/refman/8.0/en/cursors.html

Exemple avec une boucle loop (exemple 6)

```
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

https://dev.mysql.com/doc/refman/8.0/en/create-function.html

https://dev.mysql.com/doc/refman/8.0/en/faqs-stored-procs.html

https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-data-definition.html

Présentation (exemple 7)

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
     comment 'test de function'
begin
     declare sumSal float(7,2);
     select sum(sal) into sumSal
     from emp
     where job = my_job;
     return (sumSal);
end ;
//
delimiter ;
```

Remarque sur le « deterministic »

Le mot clé « **deterministic** » est obligatoire.

On met « deterministic », si le résultat est toujours le même quelles que soient les entrées (ce qui veut dire qu'il n'est pas fonction du contenu de la BD).

On met « **not deterministic** » sinon. Ici, le calcul varie en fonction du contenu de la BD.

https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

Concrètement, on met **toujours « deterministic »** pour éviter les ennuis !

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)
```

Stockage des fonctions stockées

Les fonctions peuvent être stockée dans la BD où on les crée. Elles seront accessibles à partir de la BD.

Elles peuvent aussi être stockées dans une bibliothèque partagée pour être accessibles par tous les utilisateurs, au même titre que la fonction ABS() ou la fonction CONCAT().

La syntaxe est un peu différente et décrite ici :

https://dev.mysql.com/doc/refman/5.5/en/create-function-udf.html

Gestion des erreurs

Afficher les erreurs-warnings :

```
Show warnings ;
```

Gestion des fonctions stockées

Afficher les fonctions existantes :

```
Show function status ;
```

Afficher le code d'une fonction :

```
Show create function testFonction ;
```

Supprimer une fonction :

```
Drop function insertemp ;
```

Créer une fonction :

```
delimiter //
Create function insertemp ...
   ...
end;
//
```

Fonctions stockées dans le dictionnaire des données

Tables de stockage

Les données des procédures stockées se trouvent dans les 2 tables suivantes de la BD information_schema (le dictionnaire des données).

- information_schema.**routines**
- information_schema.**parameters**

Tables de stockage

Les données des procédures stockées se trouvent dans les 2 tables suivantes de la BD information_schema (le dictionnaire des données).

- information_schema.**routines**
- information_schema.**parameters**

Afficher les procédures existantes :

```
Select  routine_name
from    information_schema.routines ;
```

Afficher le code d'une fonction :

```
Select  routine_definition
from    information_schema.routines
where   routine_name = 'testFonction';
```

Afficher le comment d'une fonction :

```
Select  routine_comment
from    information_schema.routines
where   routine_name = 'testFonction';
```

Afficher les paramètres d'une fonction :

```
select  *
from    information_schema.parameters
where   specific_name="testFonction"\G
```

Le \G permet d'avoir un affichage « page » avec une ligne par attribut : c'est plus lisible.

TP PROCEDURES ET FONCTIONS STOCKEES

0. MySQL Workbench

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

Présentation

MySQL Workbench fournit aux DBAs et aux développeurs un environnement de développement intégré pour :

- La **conception de la BD**
- Le **développement SQL** (MySQL Workbench remplace MySQL Tools : MySQL administrator et MySQL Query Browser)
- L' **administration de la BD**

Téléchargement

https://dev.mysql.com/downloads/workbench/

Télécharger : Windows (x86, 32-bit), MSI Installer

Installation

Pas de paramétrage : version complète.

Résultats de l'installation

Répertoire d'installation

C:\Program Files\MySQL\MySQL Workbench 5.2 CE (??

Fichier exécutable

C:\Program Files\MySQL\MySQL Workbench 5.2 CE\MySQLWorkbench.exe

Principes du fonctionnement

3 usages

- SQL developement
- Data Modeling
- Server Administration

SQL developement

Si le server MySQL est lancé, quand on clique sur le « local instance MySQL » ça ouvre une fenêtre qui donne accès à un « browser » avec les BD déjà enregistrées sur le serveur et à une fenêtre qui permet de passer des commandes SQL.

SQL Administration

Si le server MySQL est lancé, quand on clique sur le « local MySQL » ou sur « Server Administration, ça ouvre une fenêtre qui permet de faire de l'administration : suivre l'état du serveur, gérer les paramètres de configuration, gérer les utilisateurs, gérer les import-export de BD.

Data Modeling

- Create new EER model / add diagram : on peut créer des tables et des liaisons et enregistrer le modèle
- Open existing EER model : permet d'ouvrir un modèle précédemment enregistré
- Create EER model from existing database : permet de créer un modèle à partir d'une BD déjà enregistrée sur le serveur
- Create EER model from SQL script : permet de créer un modèle à partir d'un fichier script SQL.

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 restants 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 correspondant au nombre de jours de retard le plus grand pour ses emprunts en cours. Par exemple, s'il devait rendre un livre avant-hier et qu'il a un livre à rendre le lendemain, on renvoie « -2 » pour avant-hier.

   Si l'adhérent n'est pas en retard, on renvoie un résultat positif correspondant au nombre de jours d'emprunt restant le plus petit pour ses emprunts en cours. Par exemple, s'il doit rendre un livre demain et un autre après-demain, on renvoie « +1 » pour demain.

   (Pour ces deux derniers cas, 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).
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.** On affichera le n° et le nom de l'adhérent ainsi que le n° du livre avec titre et auteur et les dates d'emprunt et de retour. On triera le résultat par date d'emprunt décroissante et par ordre alphabétique de titre. Tester cette procédure sur plusieurs adhérents. On vérifiera que la procédure fonctionne avec des adhérents qui n'ont jamais emprunté (le 30 et le 32 par exemple).
5) Ecrire une **procédure qui affiche les exemplaires disponibles d'un titre.** On affichera toutes les caractéristiques des exemplaires. Tester la procédure sur plusieurs livres. Vérifiez le résultat pour un titre avec aucun exemplaire disponible.
6) Ecrire une **procédure qui affiche les titres d'un auteur et le nombre d'exemplaires disponibles par titre**. On testera avec LEWIS CAROLL, GILBERT HOTTOIS et kenneth white.
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.

   Pour résoudre ce problème, on écrira d'abord :

   une fonction qui dit si un livre est disponible ou pas,

   une fonction qui renvoie le nombre de livres actuellement empruntés par un adhérent.
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.

13) Ecrire une **procédure qui crée une table de bilan annuel** qui par adhérent donne le nombre d"emprunts, la durée moyenne d"emprunts, le nombre de retards, la durée moyenne de retards, la liste des styles empruntés. La table procédure remplie aussi la table.

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. Faire le graphe des tables.

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.

Le but est de produire ce résultat :

```
mysql> call bilanExam(2)\G
*************************** 1. row ***************************
            nex: 1
       note max: 13
     les élèves: 8:Chenu
       note min: 8
     les élèves: 4:David
       count(*): 12
tous les élèves: 1:Dupont, 2:Durand, 3:Dupont, 4:David, 5:Dupuis,
    6:Carlier, 8:Chenu, 9:Michelin, 10:Nerval, 11:Janset,
    12:Brulard, 13:Jordan

1 row in set (0.02 sec)
```

On a intérêt à commencer par résoudre les sous-problèmes suivants :

- Requête qui détermine toutes les meilleures notes
- Fonction qui calcule la meilleure note d'un examen
- Fonction qui calcule la moins bonne note d'un examen

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.