SQL et programmation

Responsable du module: Hamida LAGRAA

Hamida.lagraa@univ-lyon1.fr

Contenu du module: Ressource R3.07

- Les fonctions stockées
- Curseurs et Triggers
- Les transactions
- Les vues

Evaluation

- 2 TPs notés
- Bonus/Mallus sur les TPs

Partie 1

Les fonctions stockées

Fonctions stockées: C'est quoi?

- Sous programme ou module: tout programme, éventuellement paramétré, qui peut être utilisé en un plusieurs points d'un programme principal. On dit qu'il est appelé.

- 2 types de sous programmes:
  - La **fonction**: sous programme qui retourne un seul résultat. Lors de son appel, le nom de la fonction est utilisé pour désigner la valeur retournée. Une fonction peut prendre des paramètres (données) en entrée.
  - La **procédure**: sous programme qui ne rend aucun résultat (pas de return) ou en rend plusieurs via des paramètres de sorties (paramètres de type référence).

- Procédure/fonction stockée: un sous programme écrit et stocké compilé au niveau du SGBD (ou même tire que les tables).
- La procédure/fonction est compilée une fois par le SGBD, au moment de sa création, ce qui permet de l'exécuter rapidement au moment de l'appel.

Fonctions stockées: Pourquoi?

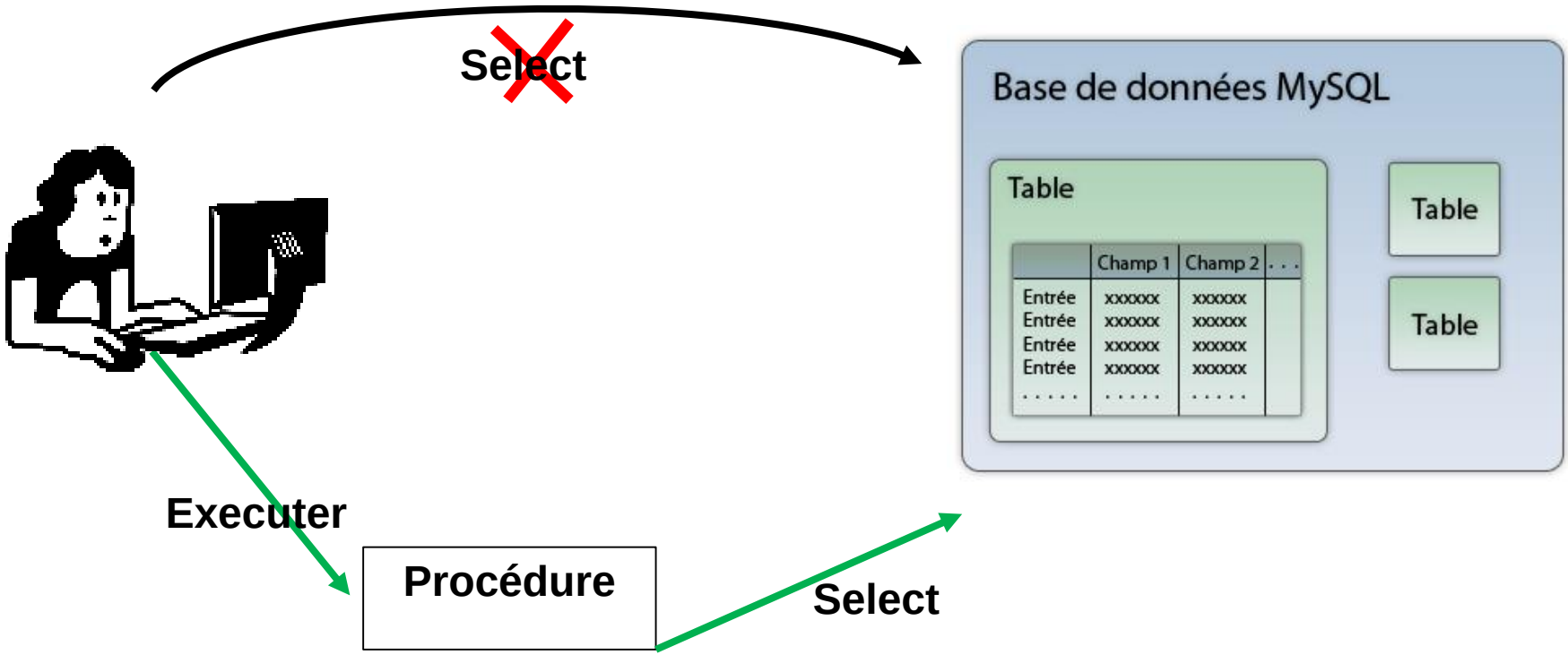
- Modularité : un sous-programme peut être réutilisable, car il peut être appelé par un autre.

- Portabilité : un sous-programme est indépendant du système d'exploitation qui héberge le serveur MySQL. En changeant de système, les applicatifs n'ont pas à être modifiées.

- Rapidité: évite les échanges réseaux qui sont nécessaires quand les mêmes fonctionnalités sont implantées dans un programme externe communiquant en mode client/serveur avec la base de données. Seul le résultat final est transmis.

- Sécurité, car les sous-programmes s'exécutent dans un environnement *a priori sécurisé* (SGBD) où il est plus facile de garder la maîtrise sur les ordres SQL exécutés et donc sur les agissements des utilisateurs.

Fonctions stockées: Pourquoi?



Fonctions Stockées

DELIMITER délimiteur
CREATE FUNCTION [*nomBase.*]*nomFonction(*
[*param typeMySQL*
[,*param2 typeMySQL*]] ...)

RETURNS *typeMySQL*
[{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT '*commentaire*']
BEGIN
[DECLARE ... ;]
bloc d'instructions SQL et MySQL ... ;
contenant un « RETURN variable ; »
END *délimiteur*

Fonctions Stockées: Exemple

```
DROP function if exists test2;
delimiter $
create function test2(x integer)returns integer
BEGIN
 DECLARE   v_somme  int default  0;
 DECLARE   v_entier int default  0;
REPEAT
 SET v_somme := v_somme + v_entier;
 SET v_entier := v_entier + 1;
UNTIL (v_entier > x)
END REPEAT;
 return v_somme;
END
$
delimiter ;
Select test2(10);
```

Procédures Stockées

DELIMITER délimiteur
CREATE PROCEDURE [*nomBase.*]*nomProcédure*(
[[IN | OUT | INOUT] *param typeMySQL*
[,[IN | OUT | INOUT] *param2 typeMySQL*]] ...)
[{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT '*commentaire*'
]
BEGIN
[DECLARE ... ;]
bloc d'instructions SQL et MySQL ... ;
END

Délimiteur

- La procédure est créée dans la base de données courante ou dans *nomBase*.
- IN : paramètre d'entrée, OUT : paramètre de sortie et INOUT: paramètre d'entrée et de sortie.
- CONTAINS SQL : la procédure interagit avec la base.
- NO SQL : la procédure n'interagit pas avec la base.
- READS SQL DATA : les interactions avec la base sont en lecture seulement.
- MODIFIES SQL DATA : des mises à jour de la base sont possibles.
- SQL SECURITY : privilèges d'execution de la procédure (privilège du créateur option par défaut : *definer-rights procedure) ou ceux de l'utilisateur qui appelle la procédure* (*invoker-rights procedure).*
- COMMENT : commenter la procédure au niveau du dictionnaire des données
- *délimiteur : délimiteur de commandes différent de « ; » (obligatoire)*.

Procédures Stockées: Exemple

```
delimiter $
create procedure test1()
BEGIN
DECLARE v_salle varchar(10);
SELECT nomSalle INTO v_salle
FROM Salle
WHERE nSalle='S01';
END
$
Delimiter ;
call test1();
```

Procédures Stockées: Exemple

```
delimiter $
create procedure test1(IN x integer, OUT s integer)
BEGIN
DECLARE i integer default 0;
Set s:=0;
While (i<x) DO
Set s:=s+i;
Set i:=i+1;
END WHILE;
END
$
Delimiter ;
Set @v:=0;
call test1(5,@v);
Select @v;
```

Suppression et appel

- **Suppression**

```
DROP PROCEDURE IF EXISTS [nomBase.]nomProcedure;
DROP FUNCTION IF EXISTS [nomBase.]nomProcedure;
```

- **Appel de procedure**

```
CALL [nomBase.]nomProcedure(listparam);
```

- Une fonction est appelée directement par son nom

Les variables

- **Declaration**

DECLARE *nomVariable1[,nomVariable2...] typeMSQL [DEFAULT expression];*

- ***Affectation***

SET nomvariable := valeur;

Instructions SQL dans un sous-programme

SELECT *col1 [,col2 ...]**INTO variable1 [,variable2 ...]***

FROM *nomTable ...;*

```
delimiter $
create procedure test1()
BEGIN
DECLARE v_salle varchar(10);
SELECT nomSalle INTO v_salle
FROM Salle
WHERE nSalle='S01';
END
$
Delimiter ;
call test1();
```

- Si la requête renvoie plusieurs valeurs , vous aurez l'erreur : « ERROR 1172 (42000): Result consisted of more than one row ».
- Si la requête ne renvoie rien , *aucune erreur n'est soulevée et la variable est* inchangée

Structures de contrôle: Instruction IF

```
IF condition THEN
  instructions;
[ELSEIF condition THEN
  instructions;]
[ELSE
  instructions;]
END IF;
```

```
delimiter $
create procedure test1()
BEGIN
DECLARE v_ename varchar(10) Default 'OSBORNE';
IF v_ename = 'OSBORNE' THEN
  SELECT "22";
END IF;
END
$
Delimiter ;
call test1();
```

Structures de contrôle: Instruction CASE

```
CASE variable
    WHEN expr1 THEN instructions1;
    WHEN expr2 THEN instructions2;
    …
    WHEN exprN THEN instructionsN;
    [ELSE instructionsN+1;]
END CASE;
```

```
CASE
WHEN condition1 THEN instructions1;
WHEN condition2 THEN instructions2;
…
WHEN conditionN THEN instructionsN;
[ELSE instructionsN+1;]
END CASE;
```

```
delimiter $
create procedure test1()
BEGIN
DECLARE v_note int Default 10;
DECLARE v_mention varchar(2);
CASE
WHEN v_note >= 16 THEN SET v_mention := 'TB';
WHEN v_note >= 14 THEN SET v_mention := 'B';
WHEN v_note >= 12 THEN SET v_mention := 'AB';
WHEN v_note >= 10 THEN SET v_mention := 'P';
ELSE SET v_mention := 'R';
END CASE;
END
$
Delimiter ;
call test2();
```

Structures de contrôle: Instruction Loop

```
[etiquette:] LOOP
  instruction(s);
  . . .
   [IF condition LEAVE etiquette ; ]
   [IF condition ITERATE etiquette;]
END LOOP [etiquette];
```

```
DROP procedure if exists test2;
delimiter $
create procedure test2()
BEGIN
 DECLARE   v_somme  int default  0;
 DECLARE   v_entier int default  0;
boucle: LOOP
 SET v_somme := v_somme + v_entier;
 SET v_entier := v_entier + 1;
 IF v_entier > 100 THEN        LEAVE boucle;
 END IF;
 END LOOP;
 Select v_somme;
END
$
```

Structures de contrôle: Instruction While

[*etiquette:*] *WHILE condition DO*
instructions;
END WHILE [*etiquette*];

```
DROP procedure if exists test2;
delimiter $
create procedure test2()
BEGIN
 DECLARE   v_somme  int default  0;
 DECLARE   v_entier int default  0;
WHILE (v_entier <= 100) DO
 SET v_somme := v_somme + v_entier;
 SET v_entier := v_entier + 1;
END WHILE;
 Select v_somme;
END
$
```

Structures de contrôle: Instruction Repeat

[etiquette:] REPEAT
instructions;
UNTIL *condition*
END REPEAT [etiquette];

```
DROP function if exists test2;
delimiter $
create function test2(x integer)returns integer
BEGIN
 DECLARE   v_somme  int default  0;
 DECLARE   v_entier int default  0;
REPEAT
 SET v_somme := v_somme + v_entier;
 SET v_entier := v_entier + 1;
UNTIL (v_entier > x)
END REPEAT;
 return v_somme;
END
$
delimeter ;
Select test2();
```

Appel d'une procédure avec paramètres de sortie

- Deux solutions:
  - Ecrire une procédure main() qui aura pour rôle d'appeler la procédure en question avec une variable.

```
delimiter $
create procedure  main()
BEGIN
 DECLARE   v default  0;
 Call proc(v);
 Select v;
END
$
```

  - Utiliser des variables de sessions: variable précédées de @. Ces variables sont détruites à la fin de votre session.

```
Set @v:=0;
Call proc(@v);
Select @v;
```