SQL & prog

Responsable du module: Hamida LAGRAA

Hamida.lagraa@univ-lyon1.fr

Partie 2

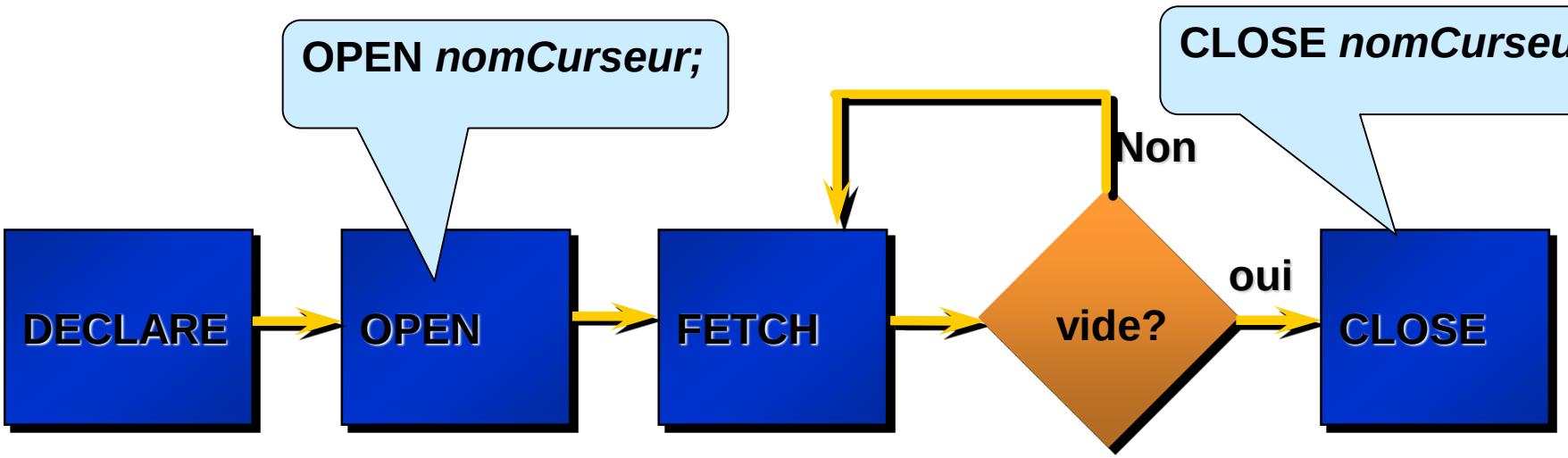
Curseurs et déclencheurs

Cursors and Triggers

Curseurs (Cursors)

- But : stocker et traiter le résultat d'un select comportant plusieurs lignes
- Tout ordre SQL utilise un curseur pour s'exécuter:
  - curseur implicite
    - ✓ tout ordre LMD
    - ✓ SELECT … INTO ...
  - curseur explicite
    - ✓ déclaré dans un module
- Le curseur reflète l'allocation d'une zone mémoire côté serveur pour le traitement des enregistrements concernés

Mise en œuvre curseur explicite



- **Déclaration requête SQL**
- **Ouverture et exécution**
- **Distribution ligne courante**
- **Teste existence ligne**
- **Libération du curseur**

FETCH *nomCurseur INTO listeVariables;*

```
DECLARE nomcurseur CURSOR FOR requête SQL;
Attention: déclaration après les variables;
```

Mise en œuvre curseur explicite : test de fin

- Deux possibilités:
  - Utiliser un compteur
  - Utiliser une exception

Curseurs: Exemple avec un compteur

```
delimiter $
create procedure testCurs()
BEGIN
DECLARE cpt int;
DECLARE v_pu, v_tot decimal;
-- déclarer le curseur
DECLARE curs CURSOR FOR
SELECT p.PU
FROM PRODUIT p
WHERE p.Qte_Stock>1;

set v_tot:=0;
-- compter le nombre de lignes du curseur
select count(*) into cpt
FROM PRODUIT p
WHERE p.Qte_Stock>1;

OPEN curs;
While cpt>0 DO
 FETCH curs INTO v_pu;
 SET v_tot := v_tot+v_pu;
 SET cpt:= cpt-1;
END While;
CLOSE curs;
SELECT v_tot AS 'Total ';
END$
delimiter ;
call testCurs();
```

Gestion des exceptions

- Exception ou erreur : Tout événement qui survient pendant l'exécution.
- Exemple :

```
SELECT * FROM x;

ERROR 1046 (3D000): No database selected
```

- Un code SQLSTATE est composé de 5 caractères et désigne une classe d'erreurs ou de retour SQL. Les deux premiers caractères :
  - 00 » signifie « succès ».
  - '01' contient tous les avertissements et correspond au mot clé SQLWARNING.
  - '02' est la classe NOT FOUND.
  - Toutes les autres classes correspondent au mot-clé SQLEXCEPTION

Gestion des exceptions

Exception ? Tout événement qui survient pendant l'exécution.

DECLARE { CONTINUE | EXIT }
HANDLER FOR
{ SQLSTATE [VALUE] '*valeur_sqlstate*' | *nomException* | *SQLWARNING*
| NOT FOUND | SQLEXCEPTION | *code_erreur_mysql* }
instructions MySQL;
[, { SQLSTATE...} ...]

- CONTINUE *force à poursuivre l'exécution de programme* lorsqu'il se passe un événement prévu dans la clause FOR.
- *EXIT fait sortir l'exécution du bloc courant (entre BEGIN et END).*
- SQLSTATE permet de couvrir toutes les erreurs d'un état donné.
- *nomException s'applique à la gestion des exceptions nommées (présentées plus loin).*
- SQLWARNING permet de couvrir toutes les erreurs d'état SQLSTATE débutant par 01.
- NOT FOUND permet de couvrir toutes les erreurs d'état SQLSTATE débutant par 02.
- SQLEXCEPTION gère toutes les erreurs qui ne sont ni gérées par SQLWARNING ni par NOT FOUND.
- *instructions MySQL : une ou plusieurs instructions du langage de MySQL (*appel possibles par CALL d'une fonction ou d'une procédure cataloguée).

Gestion des exceptions

- Prévoir l'appel d'une procedure :

```
DECLARE EXIT HANDLER FOR NOT FOUND
CALL p8765432.pasTrouve(v_salle);
```

- **Utiliser une variable**

```
DECLARE CONTINUE HANDLER FOR 1172
SET flagPlusDun :=1

IF flagPlusDun THEN
SELECT 'Il y a plusieurs …'
```

Curseurs: Exemple avec une exception

- Il faut déclarer un NOT FOUND handler qui permet de savoir si on est arrivé à la fin du curseur. En effet, à chaque fois qu'on fait appel à FETCH, le curseur tente de lire la ligne suivante dans le résultat.
- Syntaxe du NOT FOUND handler:

DECLARE CONTINUE HANDLER FOR NOT FOUND SET nomhandler := 1;

- nomhandler est une variable qui indique que le curseur a atteint la fin de l'ensemble résultat.
- La déclaration du handler doit apparaitre après celle du curseur.

Curseurs: Exemple avec une exception

```
DROP procedure if exists testCurs;
delimiter $
create procedure testCurs(IN numS varchar(11), IN typeP varchar(9))
BEGIN
DECLARE fincurs BOOLEAN DEFAULT 0; /* déclarer une variable pour l'erreur not found*/
DECLARE v_nomSeg VARCHAR(20);
DECLARE v_nomPoste VARCHAR(20);
DECLARE v_nomSalle VARCHAR(20);
DECLARE v_nPoste VARCHAR(7);
DECLARE v_tot integer default 0;
DECLARE curs CURSOR FOR
-- déclarer le curseur
SELECT p.nomPoste,p.nPoste,s.nomSalle
FROM Poste p, Salle s
WHERE p.indIP = numS AND p.typePoste = typeP
AND p.nSalle = s.nSalle;
-- Déclarer un handler pour l'erreur not found qui va positionner la variable fincurs à 1
DECLARE CONTINUE HANDLER FOR NOT FOUND SET fincurs := 1;

OPEN curs;
FETCH curs INTO v_nomPoste,v_nPoste,v_nomSalle;
While NOT fincurs DO
SET v_tot := v_tot+1;
select v_tot;
FETCH curs INTO v_nomPoste,v_nPoste,v_nomSalle;
END While;

CLOSE curs;

SELECT v_tot AS 'Total ';
END
$
Delimiter ;
call testCurs('130.120.80','UNIX');
```

Déclencheurs Bases de Données

(TRIGGER)

Généralités

- Un déclencheur est un traitement (sous forme de routine SQL) qui s'exécute automatiquement en réponse à un événement.
- Il sert à:
  - Programmer toutes les règles de gestion qui n'ont pas pu être mises en place par des contraintes au niveau des tables
  - Déporter des contraintes au niveau du serveur et alléger ainsi la programmation client.
  - Renforcer les aspects de sécurité.

Création d'un déclencheur

CREATE TRIGGER *nomDéclencheur*
{ BEFORE | AFTER } { DELETE | INSERT | UPDATE }
ON *nomTable*
FOR EACH ROW
{ *instruction;* |
[*etiquette:*] *BEGIN*
instructions;
END [*etiquette*]; }

- BEFORE | AFTER précise la chronologie entre l'action à réaliser par le déclencheur et la réalisation de l'événement.
- DELETE | INSERT | UPDATE précise la nature de l'événement
- ON *nomTable spécifie la table associée au déclencheur .*
- FOR EACH ROW différencie les déclencheurs LMD au niveau ligne (le niveau état n'est pas encore pris en charge par MySQL).

Utilisation des valeurs avant et après

Variables OLD et NEW

- Les déclencheurs de lignes peuvent accéder aux anciennes et aux nouvelles valeurs des colonnes de la ligne affectée par la mise à jour prévue par l'événement avec les identificateurs OLD et NEW.
- Selon l'événement:

Evénement	old.colonne	new.colonne
DELETE	Ancienne valeur	non
UPDATE	ancienne valeur	Nouvelle valeur
INSERT	non	Nouvelle valeur

Déclencheur : exemple

```
Delimiter $
CREATE TRIGGER Trig_APRES_Installer
AFTER DELETE ON Installer FOR EACH ROW
BEGIN
    UPDATE Poste SET nbLog=nbLog - 1 WHERE nPoste =
    OLD.nPoste;
    UPDATE Logiciel SET nbInstall = nbInstall - 1 WHERE nLog =
    OLD.nLog;
END$
Delimiter ;
```

Suppression d'un Déclencheur

DROP TRIGGER [*nomBase.]nomDéclencheur;;*