Programmation dans MySQL

1. Introduction

Il est possible d'intégrer des programmes écrits en SQL directement à l'intérieur d'une base de données. Ces programmes peuvent se substituer à des requêtes exécutées de l'extérieur de la base de données, par exemple à partir de code PHP ou d'un autre langage.

Cette technologie permet

- de limiter les flux entre le code de traitement et le serveur de base de données
- d'encapsuler de la logique dans la base de données
- de garantir l'intégrité des données avec les triggers

Le dernier point suppose que le développeur de la base de données ne fait pas confiance aux différents développeurs et met en donc place un contrôle interne à la base de données afin de la protéger contre des opérations qu'il considère délétères pour l'intégrité des données.

Ces programmes sont écrits dans le langage SQL/PSM (triggers ou procédures stockées), puis enregistrés dans la base de données. Ils peuvent ensuite être exécutés par la base de données elle-même (triggers événementiels) ou être appelés à partir d'un programme, d'un trigger ou d'une autre procédure stockée.

2. SQL/PSM

SQL : Structured Query Language (Langage d'interrogation structuré)
PSM : Persistent Stored Modules (procédures stockées)

C'est une norme universelle qui définit la structure du langage de programmation en SQL. De la même manière SQL, PSM est implémenté avec de légères différences entre les différents fournisseurs de SGBD, MySQL, SQL Server, Oracle, etc.

Principe

Comme la plupart des programmes, les programmes SQL/PSM sont structurés en blocs.

Chaque bloc peut être exécuté

- directement dans le SGBD (dans une fenêtre SQL)
- automatiquement dans un trigger
- suite à l'appel d'une procédure stockée à partir d'un programme extérieur

3. Les triggers

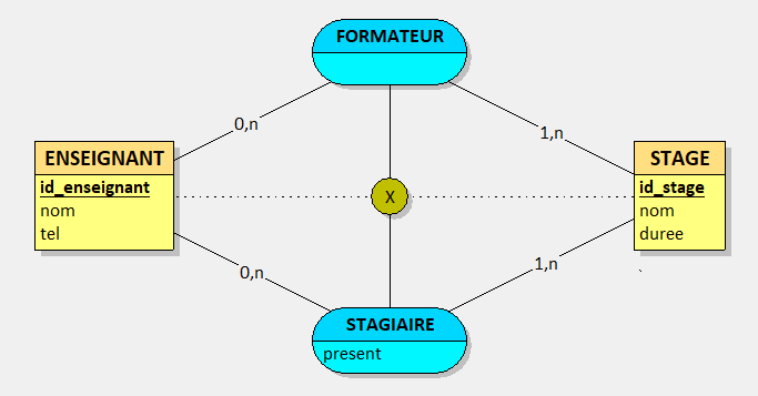
Un trigger (déclencheur) est un programme SQL intégré dans une base de données, attaché à une table précise et qui s'exécute automatiquement lors d'un événement sur cette table.

Les événements susceptibles de déclencher l'exécution du trigger sont l'*insertion*, la *modification* et la *suppression* d'enregistrements. (Les SELECT ne constituent pas des événements)

Exemple d'utilisation : ne permettre une insertion que si une condition spécifique est respectée.

Les triggers traduisent les contraintes conceptuelles, exclusion, partition, totalité, égalité, inclusion, …), mais pas uniquement. Un trigger peut également effectuer des traitements automatiques indépendant de toute contrainte (mise à jour d'un total, …)

3.1 Exemple 1 : Contrainte d'exclusion



Des stages sont proposés aux enseignants d'une école. Les enseignants peuvent suivre un stage et peuvent également en être le formateur. Ils ne peuvent évidemment pas avoir les 2 rôles en même temps. C'est le sens de la contrainte d'exclusion.

Cette situation se traduit par l'interdiction d'avoir le même couple id_enseignant + id_stage à la fois dans l'association FORMATEUR et dans l'association STAGIAIRE.

Il n'est pas possible d'exprimer cette contrainte dans la base de données (part une contrainte, un index ou autre), il faut donc faire appel à un trigger pour l'implémenter.

Ce trigger devra s'exécuter *avant l'insertion ou la modification* d'un enregistrement dans la table FORMATEUR : On devra vérifier que le couple id_enseignant + id_stage n'est pas déjà présent dans la table STAGIAIRE, *et vice versa*, *avant l'insertion ou la modification* d'un enregistrement dans la table STAGIAIRE on devra vérifier que le couple id_enseignant + id_stage n'est pas déjà présent dans la table FORMATEUR.

Codage du trigger « avant insertion » en SQL/PSM

Le trigger sera exécuté automatiquement par la base de données avant d'insérer l'enregistrement dans la table ; il pourra donc empêcher l'insertion si les conditions requises ne sont pas remplies.

```
CREATE TRIGGER insert_formateur
      BEFORE INSERT ON formateur
      FOR EACH ROW
BEGIN
      DECLARE nb INTEGER ;
      SELECT COUNT(*) INTO nb
      FROM STAGIAIRE
      WHERE id_enseignant = NEW. id_enseignant
      AND id_stage = NEW.id_stage ;
      IF (nb = 1) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = "Opération impossible" ;
      END IF ;
END ;
```

Analyse du code PSM :

CREATE TRIGGER	instruction de création d'un nouveau trigger (cf. CREATE TABLE)
BEFORE INSERT ON	sur quel événement exécuter le trigger, ici avant l'insertion
FOR EACH ROW	en cas d'insert multiple, le trigger doit s'exécuter pour chaque ligne. (La valeur FOR STATEMENT ne permet qu'une seule exécution)
BEGIN	début du programme
DECLARE nb INTEGER	déclaration d'une variable de type ENTIER
SELECT COUNT(*) INTO nb FROM STAGIAIRE WHERE id_enseignant = NEW.id_enseignant AND id_stage = NEW.id_stage	on regarde dans la table STAGIAIRE si on trouve un enregistrement avec le même id_enseignant et le même id_stage que ceux que l'on veut insérer. NEW fait référence à la nouvelle ligne sur laquelle porte l'insertion
IF (nb = 1) THEN	Instruction de branchement conditionnel (si on a trouvé un enregistrement dans la table STAGIAIRE)
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Opération impossible"	Provoque une erreur et interrompt l'insertion. On précise un numéro ainsi qu'un texte d'erreur
END IF	Fin du bloc IF
END	Fin du programme

Fonctionnement du trigger

1. Insertion dans STAGIAIRE

```
INSERT INTO STAGIAIRE(id_enseignant, id_stage, present) VALUES
(1,1,0) ;
```

2.Insertion dans FORMATEUR

```
INSERT INTO FORMATEUR(id_enseignant, id_stage) VALUES (2,2); → OK

INSERT INTO FORMATEUR(id_enseignant, id_stage) VALUES (1,2); → OK

INSERT INTO FORMATEUR(id_enseignant, id_stage) VALUES (1,2); → ERREUR
```

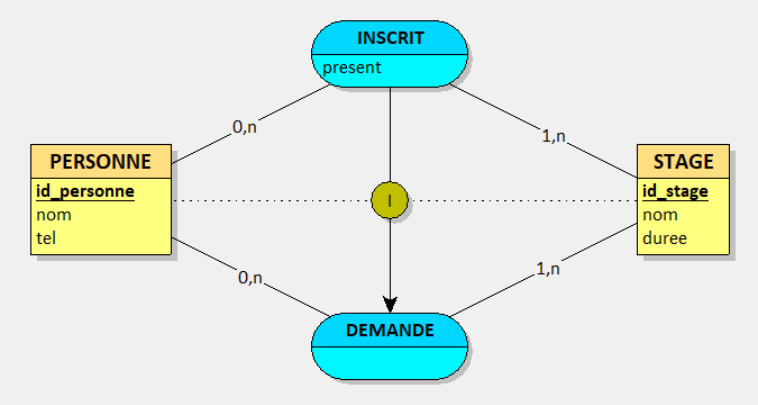
Côté application

```
< ?php
$bdd = new PDO('mysql:host=localhost;dbname=base_de_donnees', 'username', 'password ');
$req = $bdd->prepare("INSERT INTO formateur ('id_enseignant', 'id_stage') VALUES (2, 2) ;") ;
$req->execute();
echo $req->errorInfo()[2] . "<br />"
// pas de message = requête sans conflit
$req = $bdd->prepare("INSERT INTO formateur ('id_enseignant', 'id_stage') VALUES (1, 1) ;") ;
$req = $bdd->prepare("INSERT INTO formateur ('id_enseignant', 'id_stage') VALUES (2, 2) ;") ;
$req->execute();
echo $req->errorInfo()[2] . "<br />"
// « Opération impossible » (message retourné par le trigger) = conflit dans la requête.
 ?>
```

Il est donc possible de savoir à partir d'un programme extérieur si un trigger s'est exécuté sans erreur ou non.

3.2 Exemple 2 : Contrainte d'inclusion

Dans cet exemple, des personnes peuvent s'inscrire à des stages. Mais en raison du grand nombre de demandes, seules les personnes ayant déposé une demande peuvent être effectivement inscrites à un stage. C'est le sens de la contrainte d'inclusion : on a l'ensemble des demandes et l'ensemble des inscriptions est un sous-ensemble de celui des demandes :



Au niveau de la base de données, cela se traduit par le fait qu'un couple id_personne/id_stage doit se trouver dans l'association DEMANDE pour pouvoir être inséré dans l'association INSCRIT. Lors de l'insertion dans INSCRIT, on devra vérifier la présence du couple id_personne/id_stage dans DEMANDE.

Triggers associés à la contrainte :

1. Avant insertion (et avant modification) dans INSCRIT, contrôler que le couple id_personne/id_stage est présent dans DEMANDE

2. Avant suppression dans DEMANDE, contrôler que le couple id_personne/id_stage ne se trouve pas dans INSCRIT

Codage du trigger « avant insertion dans inscrit » en SQL/PSM

```
CREATE TRIGGER insertion_inscrit
      BEFORE INSERT ON inscrit
      FOR EACH ROW
BEGIN
      DECLARE nb INTEGER ;
      SELECT COUNT (*) INTO nb
      FROM demande
      WHERE id_personne = NEW.id_personne
      AND id_stage = NEW.id_stage ;
      IF (nb = 0) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = "Opération impossible"
      END IF ;
END ;
```

Codage du trigger « avant suppression sur demande » en SQL/PSM

```
CREATE TRIGGER suppression_demande
      BEFORE DELETE ON demande
      FOR EACH ROW
BEGIN
      DECLARE nb INTEGER ;
      SELECT COUNT (*) INTO nb
      FROM inscrit
      WHERE id_personne = OLD.id_personne
      AND id_stage = OLD.id_stage ;
      IF (nb = 1) THEN
            SIGNAL SQLSTATE(45000)
            SET MESSAGE_TEXT = "Opération impossible"
      END IF ;
END ;
```

Attention au mot-clé "OLD" : En effet, dans ce cas la valeur est déjà présente dans la table, c'est donc une ancienne valeur et non pas une nouvelle.

3.3 Chronologie et type de déclenchement

BEFORE/AFTER INSERT/UPDATE/DELETE ON Table

- BEFORE : Le trigger s'exécute avant l'action (il peut donc bloquer l'action)
- AFTER : Le trigger s'exécute après l'action (qui a donc lieu de toute façon)

3.4 Nombre de déclenchements

- FOR EACH ROW : Le trigger se déclenche pour chaque ligne concernée par l'opération
- FOR STATEMENT : Le trigger ne se déclenche qu'une seule fois pour l'ensemble des lignes

3.5 Récupération des valeurs

NEW, OLD : Préfixes permettant d'accéder aux valeurs

Sur insert : NEW.attribut = les valeurs que l'on est en train d'insérer

Sur update : NEW.attribut = les nouvelles valeurs, OLD.attribut = les valeurs à remplacer

Sur delete : OLD.attribut = les valeurs que l'on est en train de supprimer

3.6 Intégration dans mySQL

On commence par supprimer le trigger au cas où il existerait déjà

```
DROP TRIGGER IF EXISTS nom_du_trigger ;
```

La création du trigger doit être encadrée dans un bloc « DELIMITER », car la syntaxe de création comporte des points-virgules, en nous ne voulons pas que ceux-ci soient interprétés comme des fins d'instruction au moment de la création :

```
DELIMITER //
CREATE TRIGGER nom_du_trigger
...
BEGIN
...
END;
// DELIMITER ;
```

4. Syntaxe SQL/PSM

4.1 Casse

Il n'y a pas de contrôle de casse en SQL/PSM : if = IF = If. Cependant, c'est toujours une bonne idée de se fixer une charte.

4.2 Indentation

Il n'y a pas de contrôle de l'indentation, mais évidemment l'usage d'une bonne indentation est plus que conseillée !

Chaque instruction de termine par un point-virgule

4.3 Variables

Les déclarations de variables doivent être faites dans un bloc DECLARE, qui peut contenir autant de déclarations que nécessaire :

```
DECLARE
      nom_variable_1 type_variable_1 ;
      nom_variable_2 type_variable_2 ;
      nom_variable_3 type_variable_3 ;
      …
```

Les types de variables sont ceux du SGBD. (En MySQL : INT, CHAR(5), VARCHAR(50), …)

4.4 Blocs

Les blocs exécutables sont délimités par les mots-clé BEGIN et END ;

```
BEGIN
      instruction … ;
      …

End ;
```

4.5 Commentaires

Les commentaires sur une ligne commencent par --. Les commentaires multilignes doivent être encadrés par /* */

```
-- commentaire simple
/* commentaire
multiligne */
```

4.6 Affectations

Les affectations peuvent être effectuées au moment de la déclaration sous la forme

```
DECLARE nom_de_la_variable type_de_la_variable DEFAULT
valeur_de_la_variable ;
```

Pour affecter une valeur en cours de programme, il faut utiliser la syntaxe

```
SET nom_de_la_variable := valeur_de_la_variable ;
```

4.7 Structures conditionnelles :

4.7.1 IF

```
IF (condition) THEN
      instructions ;
      …
```

```
ELSE
      instructions ;
      …
END IF ;
```

4.7.2 CASE

```
CASE (variable)
      WHEN valeur_1 THEN action_1 ;
      WHEN valeur_2 THEN action_2 ;
      ...
      ELSE action_par_defaut ;
END CASE ;
```

4.7.3 Tests dans les conditions

```
=     égal à
!=    différent de
>=    plus grand ou égal
<=    plus petit ou égal
>     strictement plus grand
<     strictement plus petit
```

4.8 Boucles

4.8.1 WHILE

```
WHILE (condition) DO
      instructions ;
END WHILE
```

4.8.2 REPEAT

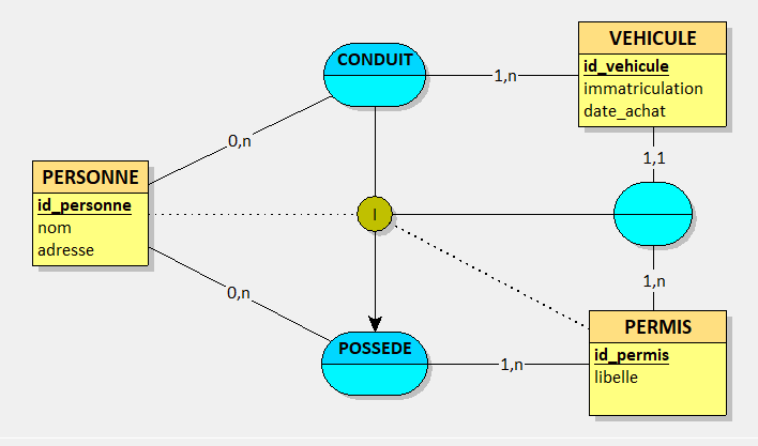
```
REPEAT
      instructions ;
UNTIL (condition) END REPEAT ;
```

4.8.3 FOR LOOP

```
FOR variable IN début..fin LOOP
      instructions ;
END LOOP ;
```

5. Exemples

5.1 Exemple 1



Les personnes peuvent posséder (association POSSEDE) un ou plusieurs permis (A, B, C, …).

Un spécifique particulier est nécessaire pour pouvoir conduire un véhicule donné (association VEHICULE/PERMIS).

Une personne peut conduire un véhicule (association CONDUIRE), mais uniquement si elle possède le permis correspondant au véhicule : c'est ce qu'indique la contrainte d'inclusion.

Cette contrainte spécifie que pour insérer une ligne dans CONDUIRE, il faut qu'il y ait une ligne dans POSSEDE avec la bonne personne et le bon permis.

Traduction :

Pour insérer un couple id_personne/id_vehicule dans conduire, on doit pouvoir trouver un couple id_personne/id_permis dans POSSEDE avec le PERMIS associé au VEHICULE du couple id_personne/id_vehicule.

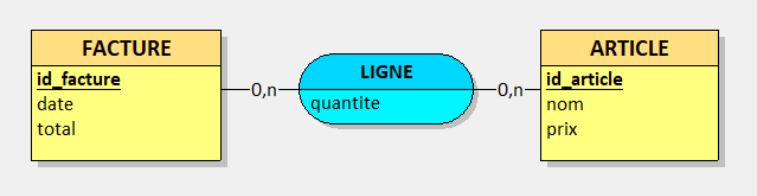
Les associations CONDUIT et POSSEDE ne contiennent pas les mêmes identifiants (id_personne/id_vehicule dans CONDUIT et id_personne/id_permis dans POSSEDE). Il faudra donc récupérer pas une sous-requête le permis associé au véhicule concerné pour pouvoir effectuer la vérification dans POSSEDE.

Code du trigger « avant insertion dans CONDUIT)

```
CREATE TRIGGER insertion_conduit
      BEFORE INSERT ON conduit
      FOR EACH ROW
BEGIN
      DECLARE nb INTEGER ;
      SELECT COUNT(*) INTO nb
      FROM possede
      WHERE id_personne = NEW.id_personne
      AND id_permis = (SELECT id_permis FROM vehicule
      WHERE id_vehicule = NEW.id_vehicule) ;
      IF (nb = 0) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Pas de permis pour ce véhicule'
      END IF ;
END ;
```

5.2 Exemple 2

Dans cet exemple, le trigger ne correspond pas à une contrainte. Il sera exécuté après l'insertion



Une facture peut concerner un ou plusieurs articles. La propriété total de FACTURE devra porter la somme des montants (ARTICLE.prix * quantite) de chaque ligne de la facture. Cette propriété permettra d'éviter de refaire le calcul à chaque manipulation/édition de la facture. Elle permettra également de conserver le total de la facture même en cas d'évolution des prix des articles. Elle devra donc être mise à jour automatiquement, directement dans la base de données, et sans laisser au programmeur le soin de la recalculer à chaque modification.

Rôle du trigger : à chaque ajout, suppression ou même modification dans LIGNE, mettre à jour le total de la facture

Code du trigger « après insertion dans LIGNE)

```
CREATE TRIGGER insertion_ligne
      AFTER INSERT ON ligne
      FOR EACH ROW
BEGIN
      UPDATE facture
      SET total = total +
            ((SELECT prix FROM ARTICLE
            WHERE id_article = NEW.id_article) * NEW.quantite)
      WHERE id_facture = NEW.id_facture ;
END ;
```

Code du trigger « après delete dans LIGNE)

```
CREATE TRIGGER suppression_ligne
      AFTER DELETE ON ligne
      FOR EACH ROW
BEGIN
      UPDATE facture
      SET total = total -
            ((SELECT prix FROM ARTICLE
            WHERE id_article = OLD.id_article) * OLD.quantite)
      WHERE id_facture = OLD.id_facture ;
END ;
```

6. Procédures stockées

Une procédure (ou fonction) stockée est un programme SQL/PSM intégré dans une base de données qui doit être appelé pour s'exécuter.

Elle permet d'enregistrer des traitements directement dans la base de données et donc

- de limiter les flux entre le serveur d'application et le serveur de données
- d'encapsuler les traitements (boîte noire)
- de réutiliser les traitements

Une procédure stockée peut être appelée

- par une autre procédure stockée
- par un trigger (permet d'optimiser les traitements identiques)
- par un programme extérieur à la base de données

6.1 Exemple de problème potentiel

Sur la base du trigger qui empêchait un enseignant d'être en même temps formateur et stagiaire dans un stage :

Nous avons besoin de 4 triggers :

- BEFORE INSERT ON stagiaire
- BEFORE UPDATE ON stagiaire
- BEFORE INSERT ON formateur
- BEFORE UPDATE ON formateur

SI l'on regarde juste la table formateur, on a 2 triggers dont le code sera identique, ce qui n'est pas du tout optimisé (risque de divergence du code).

On peut éviter cela en écrivant une procédure stockée unique qui sera ensuite appelée par les 2 triggers.

Rappel du code du trigger BEFORE INSERT ON formateur :

```
CREATE TRIGGER insert_formateur
      BEFORE INSERT ON formateur
      FOR EACH ROW
BEGIN
      DECLARE nb INTEGER ;
      SELECT COUNT(*) INTO nb
      FROM STAGIAIRE
      WHERE id_enseignant = NEW.id_enseignant
      AND id_stage = NEW.id_stage ;
      IF (nb = 1) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = "Opération impossible" ;
      END IF ;
END ;
```

On ne peut pas directement coller ce code dans une procédure stockée, car nous avons utilisé l'objet NEW qui n'existe que pendant l'insertion, dans le code d'un trigger. Nous devrons donc utiliser des paramètres pour transmettre ces valeurs à la procédure stockée.

Code de la procédure stockée

```
CREATE PROCEDURE exec_formateur_stage (IN id_enseignant INTEGER, IN
id_stage INTEGER)
BEGIN
      DECLARE nb INTEGER ;
      SELECT COUNT(*) INTO nb
      FROM STAGIAIRE
      WHERE id_enseignant = id_enseignant
      AND id_stage = id_stage ;
      IF (nb = 1) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = "Opération impossible" ;
      END IF ;
END ;
```

Nouveau code du 2 trigger sur formateur :

```
CREATE TRIGGER insert_formateur
      BEFORE INSERT ON formateur
      FOR EACH ROW
BEGIN
      CALL exec_formateur_stage(NEW.id_enseignant, NEW.id_stage) ;
END;


CREATE TRIGGER update_formateur
      BEFORE UPDATE ON formateur
      FOR EACH ROW
BEGIN
      CALL exec_formateur_stage(NEW.id_enseignant, NEW.id_stage) ;
END;
```

7. Fonctions stockées

Le principe des fonctions stockées est le même que celui des procédures stockées, à la différence près que les fonctions stockées retournent une valeur.

Sur la base de l'exemple précédent, on va essayer de gérer les erreurs potentielles directement dans e trigger afin de pouvoir les personnaliser selon la situation : « insertion impossible » ou « modification impossible ».

Code de la fonction stockée

```
CREATE FUNCTION exec_formateur_stage (IN id_enseignant INTEGER, IN
id_stage INTEGER) RETURNS INTEGER
READS SQL DATA
BEGIN
      DECLARE nb INTEGER ;
      SELECT COUNT(*) INTO nb
      FROM STAGIAIRE
      WHERE id_enseignant = id_enseignant
      AND id_stage = id_stage ;
      RETURN nb ;
```

```
END ;
```

Nouveau code du trigger BEFORE INSERT sur formateur :

```
CREATE TRIGGER insert_formateur
      BEFORE INSERT ON formateur
      FOR EACH ROW
BEGIN
      DECLARE nb INTEGER ;
      SET nb := exec_formateur_stage(NEW.id_enseignant,
            NEW.id_stage);
      IF (nb = 1) THEN
            SIGNAL SQLSTATE '45000'
            MESSAGE_TEXT = "insertion impossible"
      END IF ;
END ;
```

On utilisera exactement la même logique pour le trigger de mise à jour, en changeant uniquement le message.