MySQL Les déclencheurs (triggers)

1. Définition

On appelle trigger ou déclencheur un traitement déclenché automatiquement suite à une opération (insert, update, delete) sur une table.

Sur MySQL un trigger :

- Est associé à une et une seule table.
- Est associé à une **seule opération (insert, update ou delete).**
- Peut se déclencher **avant l'opération** (before) ou **après l'opération** (after).
- Traite les enregistrements concernés par l'opération **un par un** (for each row)
- Est exécuté dans la même transaction que l'opération.
- Ne peut pas utiliser les instructions sur les transactions.
- **Ne peut accéder qu'en lecture sur la table associée** (la table est verrouillée par le trigger).

2. Instructions sur les déclencheurs

Création d'un déclencheur

```
create trigger nomTrigger before | after  { insert |  update |  delete} on { table | view }
for each row
   begin
      instruction Sql;
   end
```

for each row permet de parcourir chaque enregistrement concerné. Le trigger travaille donc sur un seul enregistrement à la fois.
L'accès à cet enregistrement s'effectue à l'aide des préfixes **old** et **new**
old est une structure qui contient les anciennes valeurs de l'enregistrement
new est une structure qui contient les nouvelles valeur de l'enregistrement
begin et **end** sont nécessaires si le trigger comporte plus d'une instruction.

On utilise un **trigger after** pour ajouter ou mettre à jour des enregistrements (champ calculé) d'une autre table.
On utilise un **trigger before** pour effectuer des contrôles ou pour initialiser certaines données de la table liée au trigger.
Si un contrôle n'est pas respecté le trigger renvoie une erreur par l'instruction signal ce qui arrête l'opération.

On doit définir un caractère ou un ensemble de caractères pour délimiter un trigger.
L'instruction delimiter permet de préciser ce ou ces caractères.
En règle générale on utilise $$:

```
delimiter $$
        create trigger ….
$$
        create trigger ….
$$
```

Suppression d'un déclencheur

```
drop trigger [if exists] nomTrigger
```

Il n'est pas possible de désactiver un trigger sous MYSQL.

3. Principe général de fonctionnement

On accède individuellement aux enregistrements concernés grâce à la clause **for each row**
Deux structures permettent d'accéder aux valeurs ajoutées, modifiées ou supprimées de chaque colonne

- La structure **new** qui contient les nouvelles valeurs des colonnes.

  Elle est accessible en lecture écriture pour un trigger de type before portant sur l'ajout ou la modification.
  Pour changer la valeur avant son enregistrement : **set new.nomChamp = valeur**

- La structure **old** qui contient les anciennes valeurs des colonnes.

  Elle concerne uniquement les trigger portant sur la suppression ou la modification
  Elle est accessible en lecture seule quel que soit le type du trigger (after ou insert)

Avec un déclencheur de type before, il est possible d'annuler l'exécution de l'opération ayant déclenché le trigger en déclenchant une erreur à l'aide de l'instruction SIGNAL qui met fin à la transaction

```
SIGNAL sqlstate '45000' set message_text = 'message'
```

Lorsqu'une instruction Insert est lancée les traitements suivant s'enchainent :
- Déclenchement du trigger before
  - Les valeurs de l'enregistrement à ajouter sont **accessibles en lecture écriture** à l'aide de la structure new
  - Si on doit changer une valeur set new.colonne = valeur
- Vérification des contraintes
  - null, primary key, unique, foreign key, check
- Ajout dans la table
- Déclenchement du trigger after
  - Les valeurs de l'enregistrement ajouté sont **accessibles en lecture** à l'aide de la structure new

Lorsqu'une instruction Delete est lancée les traitements suivants s'enchainent :
- Déclenchement du trigger before
  - Les valeurs de l'enregistrement à supprimer sont **accessibles en lecture** à l'aide de la structure old
- Vérification des contraintes
  - foreign key
- Suppression dans la table
- Déclenchement du trigger after
  - Les valeurs de l'enregistrement supprimé sont **accessibles en lecture** à l'aide de la structure old

Lorsqu'une instruction Update est lancée les traitements suivants s'enchainent :
- Déclenchement du trigger before
  - Les nouvelles valeurs de l'enregistrement à modifier sont **accessibles en lecture écriture** à l'aide de la structure new. Elles sont donc encore modifiables
  - Les anciennes valeurs de l'enregistrement à modifier sont **accessibles en lecture** à l'aide de la structure new.
  - La comparaison permet de détecter une modification des valeurs
- Vérification des contraintes
  - null, primary key, unique, foreign key, check
- Modification dans la table
- Déclenchement du trigger after
  - Les valeurs de l'enregistrement modifié sont **accessibles en lecture** à l'aide des structures new et old

4. Utilisation pour mettre en place un système de traçabilité des opérations

On peut utiliser un déclencheur pour enregistrer automatiquement des informations de gestion. Par exemple, sur la modification d'enregistrements on enregistre la date et l'heure de la modification et l'auteur (user) de cette modification :

```
create trigger avantModificationNomTable before update on nomTable
for each row
    begin
        set new.modifiePar = CURRENT_USER();
        set new.modifieLe = now();
    end
```

Sur la suppression d'enregistrements on assure l'archivage des enregistrements supprimés en les plaçant dans une autre table.

```
create trigger avantSupressionNomTable before delete on nomTable
for each row
        insert into nomArchive (id, …, supprimeLe, supprimePar)
        values (old.id, …, now(), CURRENT_USER());
```

5. Utilisation pour interdire certaines opérations

On peut utiliser un déclencheur pour interdire certaines opérations
Par exemple on veut interdire la modification sur une table.

```
create trigger avantModificationNomTable before update on nomTable
for each row
        SIGNAL sqlstate '45000' set message_text = 'opération interdite ';
```

Pour interdire la modification d'une colonne (par exemple la colonne id), il suffit de comparer l'ancienne valeur et la nouvelle valeur.

```
create trigger avantModificationNomTable before update on nomTable
for each row
    if old.id <> new.id then
        SIGNAL sqlstate '45000' set message_text = 'L''identifiant ne peut pas être modifié';
    end if
```

6. Utilisation pour mettre en place une contrainte d'exclusion entre deux colonnes d'une table

La contrainte d'exclusion doit vérifier que lors d'un ajout ou d'une modification dans tableA, seul une des deux colonnes x ou y est renseignée.
Exemple : une commande provient d'un service (colonne x) ou d'un client (colonne y)

```
create trigger nom before insert on tableA
    if (new.x is not null and new.y is not null)  or (new.x is null and new.y is null) then
            SIGNAL sqlstate '45000' set message_text = 'contrainte d''exclusion non respectée ';
    end if;
```

Le déclencheur associé à l'événement update est identique

7. Utilisation pour mettre en place une contrainte d'exclusion entre deux tables

La contrainte d'exclusion doit vérifier que lors d'un ajout ou d'une modification dans tableA, on n'utilise pas une valeur de clé (colonne id) déjà présente dans tableB et inversement.
Exemple : Parmi les salariés on distingue les contremaîtres (tableA) et les installateurs (tableB).

```
create trigger nom before insert on tableA
for each row
    if exists (select 1 from tableB where id = new.id)  then
            SIGNAL sqlstate '45000' set message_text = 'Violation de la contrainte d"exclusion' ;
    end if;
```

Le déclencheur associé à la modification est identique mais la vérification ne sera lancée que si la colonne id a été modifiée.

```
create trigger nom before update on tableA
for each row
    if old.id != new.id then
        if exists (select 1 from tableB where id = new.id)  then
            SIGNAL sqlstate '45000' set message_text = 'Violation de la contrainte d"exclusion' ;
        end if;
    end if
```

Du côté de la tableB on retrouve les mêmes déclencheurs, il suffit d'inverser tableB et tableA

8. Utilisation pour mettre en place une contrainte check

La contrainte check sur les anciennes versions (5 et < 8.0.16) n'étant pas prise en compte sous MySQL, il faut dans ce cas utiliser deux déclencheurs pour la mettre en œuvre.

Exemple : tableA(id, borneMin, borneMax)
Contrainte : borneMax < borneMin

```
create trigger avantAjoutTableA before insert on tableA
for each row
    if new.borneMin >= new.borneMax then
        SIGNAL sqlstate '45000' set message_text = 'Intervale invalide ';
    end if;
```

Le déclencheur associé à l'événement update est identique

9. Utilisation pour mettre en place une contrainte portant sur plusieurs tables

Exemple : soit deux tables tableA(id, …, idB, montant) et tableB(id, montantMax, …)
Soit la règle : montant <= montantMax correspondant (tableA.idB = TableB.id)

```
create trigger avantAjoutTableA before insert on tableA
for each row
  if exists (select 1 from tableB where new.idB = tableB.id and  new.montant > montantMax) then
            SIGNAL sqlstate '45000' set message_text = 'Montant non valide ';
  end if;
```

Le déclencheur associé à l'événement update est identique

Ici on sait que la requête ne peut au mieux retourner qu'une seule ligne, la condition new.idB = tableB.id filtrant une seule ligne. Il est donc possible de proposer une autre version du déclencheur qui récupère la valeur de montantMax dans une variable. Il reste alors à comparer la valeur récupérée à new.Montant. Pour récupérer la valeur de montantMax on utilise une requête 'select … **into**'.

```
create trigger avantAjoutTableA before insert on tableA
for each row
    begin
        declare montantMax decimal(6,2);
        select montantMax into montantMax from tableB where tableB.id  = new.idB;
        if new.montant > montantMax then
            SIGNAL sqlstate '45000' set message_text = 'Montant non valide ';
        end if;
    end
```

Il faut surtout n'oublier aucun cas, la règle utilisant deux tables, il faut aussi la vérifier lorsqu'une modification intervient sur la seconde table (ici tableB)

```
create trigger avantModificationTableB before update on tableB
for each row
    if exists (select 1 from tableA where new.id = TableA.idB
                                    and montant > montantMax) then
            SIGNAL sqlstate '45000' set message_text = 'MontantMax invalide ';
    end if;
```

Il est possible de choisir une autre option pour que la règle soit toujours vérifiée :
Si la nouvelle valeur de la colonne montantMax est plus petite que l'ancienne alors on plafonne tous les montants supérieurs à montantMax.

```
create trigger avantModificationTableB before update on tableB
for each row
    if new.montantMax < old.montantMax then
        update tableA
            set montant = new.montantMax
        where tableA.idB = new.id
        and montant > new.montantMax;
    end if;
```

Pour éviter de lancer l'instruction update pour rien, il est possible de vérifier qu'il existe au moins un montant qui doit être plafonné suite à la modification de montantMax

```
create trigger avantModificationTableB before update on tableB
for each row
    if new.montantMax < old.montantMax then
        if exists (select 1 from tableA where new.id = tableA.idB
                    and montant >  new.montantMax) then
            update tableA
                set montant = new.montantMax
            where tableA.idB = new.id
            and montant > new.montantMax;
        end if;
    end if;
```

10. Utilisation pour maintenir automatiquement à jour un champ calculé de type Count

Exemple : soit deux tables Classe(id, libelle, nb, nbMax) et Eleve(id, nomPrenom, idClasse)

nb représente le nombre d'élèves dans la classe.
La valeur de nb par défaut vaut 0.
Si des enregistrements sont déjà présents dans les tables, on peut initialiser la valeur de nb à l'aide de la requête suivante :

```
update Classe
    set nb = (select count(*) from Eleve
              where Eleve.idClasse = Classe.id)
where Classe.id in (select idClasse from Eleve);
```

Après l'ajout d'un enregistrement dans Eleve il faut incrémenter nb

```
create trigger apresAjoutEleve after insert on Eleve
for each row
    update Classe
       set nb = nb + 1
     where Classe.id = new.idClasse
```

Lors de la suppression d'un enregistrement dans Eleve il faut décrémenter nb

```
create trigger apresSuppressionEleve after delete on Eleve
for each row
    update Classe
         set nb = nb – 1
      where Classe.id = old.idClasse
```

En cas de modification, il n'y a rien à faire si idClasse n'a pas été modifié. Sinon on incrémente nb pour la nouvelle valeur de idClasse et on décrémente nb pour l'ancienne valeur de idClasse

```
create trigger apresModificationEleve after update on Eleve
for each row
    if new.idClasse <> old.idClasse then
        update Classe
            set nb = nb + 1
        where Classe.id = new.idClasse;
        update Classe
            set nb = nb – 1
        where Classe.id = old.idClasse;
    end if;
```

Il arrive souvent que l'opération de modification sur la clé (ici idClasse) soit tout simplement interdite.

```
create trigger avantModificationEleve before update on Eleve
for each row
    if new.idClasse != old.idClasse then
        SIGNAL sqlstate '45000' set message_text = 'La modification n''est pas autorisée';
    end if;
```

Si le champ calculé 'nb' ne doit pas dépasser la valeur de nbMax (effectif limité à), il suffit de mettre en place une contrainte check ou son équivalent sous forme de trigger.

```
Alter table Classe add check (nb <= nbMax)
```

```
Create trigger avantModificationClasse before update on Classe
foreach row
    if  new.nb != old.nb or new.nbMax != old.nbMax then
        if  new.nb > new.nbMax 0 then
            SIGNAL sqlstate '45000' set message_text = 'Plus de place disponible dans cette classe';
        end if;
    end if;
```

La modification d'un champ calculé ne devrait pouvoir se faire qu'à partir des triggers

Comment empêcher un utilisateur qui accède directement à la base de données à l'aide d'une outil comme MySQLWorkbench de modifier le champs nb ?

On peut restreindre ses droits et ne pas l'autoriser à modifier le champ (on l'autorise pour les autres champs)

```
Grant update(id, libelle, nbMax) on nomBase.Classe to nomUser@nomServeur;
```

Pour l'interdire à un administrateur ayant tous les droits, on peut utiliser une variable utilisateur qui jouera le rôle d'identification (comme pour une grille de connexion)
Cette variable est initialisée dans les triggers de gestion du champ calculé.

Un trigger lancé avant la modification sur la Classe interdit la modification du champ calculé si la variable utilisateur n'est pas défini ou ne contient pas la valeur autorisant la modification du champ calculé.

```
create trigger apresAjoutEleve after insert on Eleve
for each row
begin
    set @trigger = 1;      // je signale que la demande provient d'un trigger
    update Classe
       set nb = nb + 1
     where Classe.id = new.idClasse
    set @trigger = null  // je désactive ma permission car la variable utilisateur existe toujours
end
```

```
Create trigger avantModificationClasse before update on Classe
foreach row
    if  new.nb != old.nb and (@trigger is null or @trigger != 1) then
            SIGNAL sqlstate '45000' set message_text = 'Le champ nb n'est pas modifiable';
    end if;
```

Bien évidemment cette protection est 'très limitée' tout utilisateur ayant des droits suffisants peut mettre à jour le champ calculé s'il connait le nom et la valeur de la variable utilisateur réalisant la protection.

11. Utilisation pour maintenir automatiquement à jour un champ calculé de type sum

Exemple : soit les deux tables Facture(id, date, total) et DetailFacture(idFacture idProduit, qte, montant)

total représente la somme de la colonne montant dans DetailFacture correspondant à la valeur de id :

```
Select id, sum(montant)
From Facture Join DetailFacture On Facture.id = DetailFacture.idFacture
Group By Facture.id
```

Exemple : une table client (A) avec un champ total et une table commande (B) avec le champ montant. Lors de l'ajout dans DetailFacture il faut ajouter la valeur de montant dans total (pour chaque enregistrement)

```
create trigger ajoutDetailFacture after insert on DetailFacture
for each row
    update Facture
        set total = total + new.montant
    where Facture.id = new.idFacture;
```

Lors de la suppression d'un enregistrement dans DetailFacture il faut retirer la valeur de montant à total

```
create trigger suppressionDetailFacture after delete on DetailFacture
for each row
    update Facture
        set total = total – old.montant
    where Facture.id = old.idFacture;
```

En cas de modification, on augmente total pour la nouvelle valeur de idFacture et on diminue total pour l'ancienne valeur de idFacture.
Si idFacture n'a pas été modifié, on peut lancer une seule instruction update

```
create trigger modificationDetailFacture after update on DetailFacture
for each row
    if new.idFacture <> old.idFacture then
        update Facture
            set total = total + new.montant
        where Facture.id = new.idFacture;
        update Facture
            set total = total – old.montant
        where Facture.id = old.idFacture;
    else
        update Facture
            set total = total + new.montant – old.montant
        where Facture.id = new.idFacture;
    end if;
```

Si on doit limiter la valeur du champ calculé et/ou si on souhaite protéger ce champ des modifications hors trigger, on procède de la même manière que pour un champ calculé de type count.

12. Récupération du message d'erreur retourné par le trigger dans un langage de programmation

Si on exécute une requête dans un programme, il est possible de récupérer le message retourné par le trigger en utilisant la gestion d'erreur disponible sur le langage hôte

Par exemple dans le langage Php

Soit la requête suivante lancée à partir d'un appel Ajax

```
<?php
require '../../class/class.database.php';
$db = Database::getInstance();
$nomCourse = strtoupper(trim($_POST["nom"]));
$dateCourse = $_POST["date"];
$distance = $_POST["distance"];
$sql = <<<EOD
        insert into course(dateCourse, nomCourse, distance)
        values(:dateCourse, :nomCourse, :distance);
EOD;
$curseur = $db->prepare($sql);
$curseur->bindParam(':dateCourse', $dateCourse);
$curseur->bindParam(':nomCourse', $nomCourse);
$curseur->bindParam(':distance', $distance);
try {
    $curseur->execute();
    echo $db->lastInsertId();
} catch(Exception $e) {  // le message commence par SQLSTATE[45000]: <>: 1644 #
    echo substr($e->getMessage(),strrpos($e->getMessage(), '#') + 1);
}
```

En commençant les messages d'erreur par #, il devient facilement possible de retirer le début du message d'erreur renvoyé par le serveur SQL.

```
function ajouter() {
    erreur.innerHTML = '';
    $.ajax({
            url: "ajax/ajouter.php",
            type: 'post',
            dataType: "json",
            data: { nom: nom.value, date: date.value, distance: zdlDistance.value},
            success: function (data) {
                let option = { message: "Course ajoutée", type: 'success', position : 'centerLeft'}
                Std.afficherMessage(option);
                 nom.value = '';
            },
            error: function(request) {
                erreur.innerHTML = Std.genererMessage(request.responseText, 'rouge');
            }
        }
    );
}
```