Triggers

MySQL

version 1.3

Trigger

- Les déclencheurs (triggers) n'existent que depuis la version 5 de MySQL
- Une table (ou vue) peut «héberger» 0 ou plusieurs déclencheurs. Le déclencheur est lié à une table contrairement aux procédures ou fonctions
- La différence majeure avec une procédure ou une fonction est que l'appelle n'est pas explicite (pas de CALL par exemple).
- C'est l'évènement qui appelle automatique le déclencheur.

Trigger

- On peut définir en théorie trois grandes catégories de trigger :
  - Les triggers DDL
    (Data Definition Language -> CREATE, ALTER, DROP)
  - Les triggers DML
    (Data Manipulation Language -> INSERT, UPDATE, DELETE)
  - Les triggers de remplacement
    (Permettre de remplacer l'instruction déclenchante par une autre action. Exemple : Arrêt de la base de données)

ATTENTION!

En pratique MySQL ne permet d'implémenter que les déclencheurs DML

A quoi sert un Trigger

- Les triggers permettent d'affiner les problèmes d'intégrité, c'est-à-dire la vérification de la cohérence des données saisies en elle-même ou par rapport à des données déjà présentes.

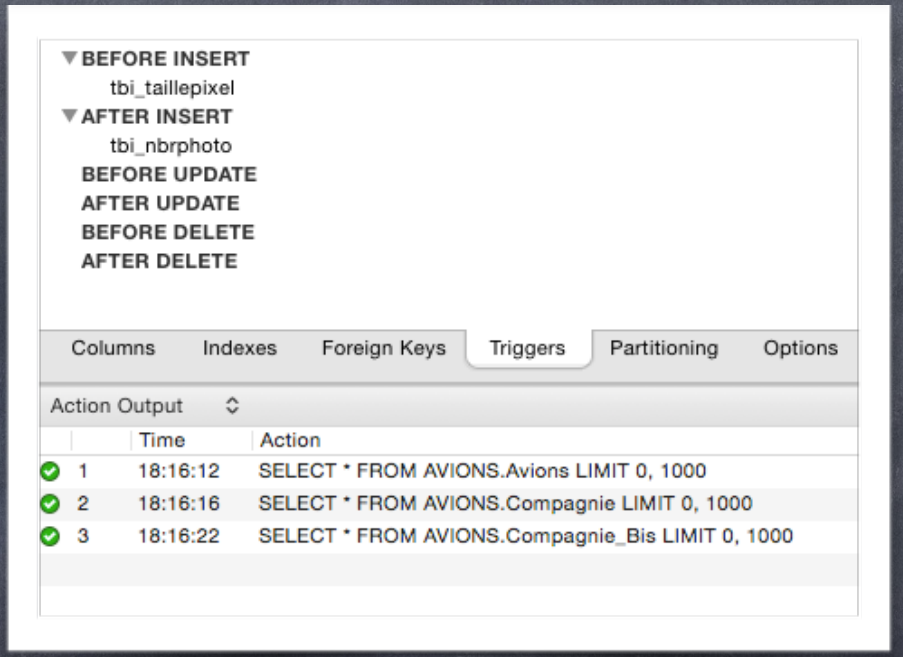
A quoi sert un Trigger

- Pour les RG qui n'ont pas pu être mise en place par des contraintes au niveau des tables.
  Exemple : il faut avoir 60 h de vol les deux derniers mois pour pouvoir prendre un avion long courrier.
- Déporter les contraintes au niveau du serveur (alléger la programmation client)
- Renforcer les aspects de sécurité et d'audit.
- Programmer l'intégrité référentielle et la réplication dans les architectures distribuées.

Mecanisme Trigger

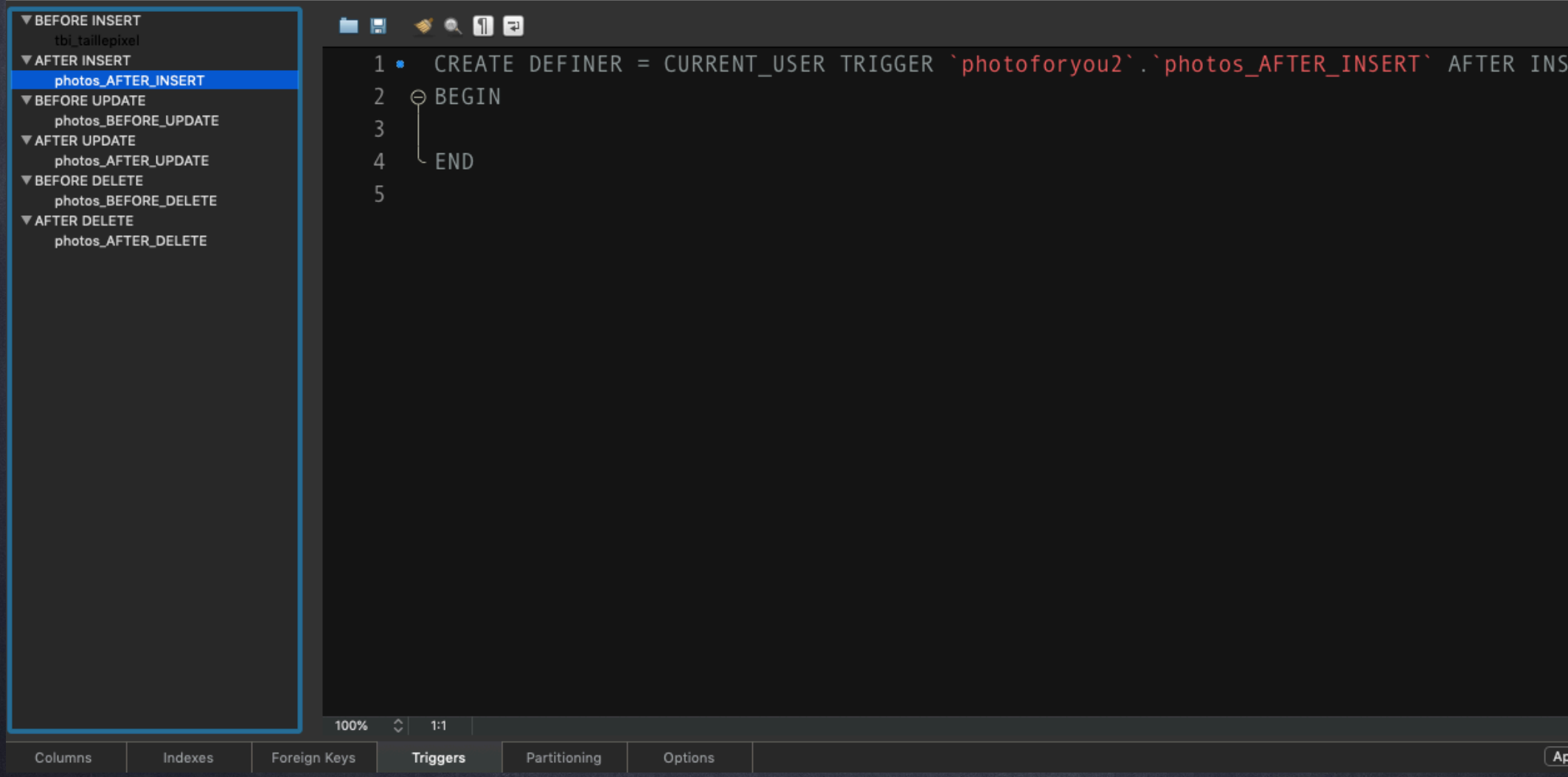
- Pour connaitre les triggers d'une base de données:
  SHOW TRIGGERS;

Avec MySQLWorkBench



- La syntaxe est composée de deux parties :
  -La description de l'évènement traqué
  -L'action à réaliser lorsque l'évènement se produit

Mecanisme Trigger



Trigger : BEFORE

Un trigger BEFORE est un trigger qui se déclenche avant l'action du DML

(INSERT,UPDATE, DELETE).

Il permet par exemple de :
Interdire l'action du DML en cas d'incohérence des données saisies : on peut gérer les valeurs limites avec un trigger. La vérification se fait avant d'avoir enregistré les nouvelles données.
Le trigger peut alors soit interdire la modification, soit la modifier, soit la laisser s'exécuter mais envoyer un message d'avertissement.
On peut aussi reformater les données saisies : on peut passer du texte en majuscules.

Trigger : AFTER

Un trigger AFTER est un trigger qui se déclenche après une action du DML : INSERT, UPDATE ou DELETE. Il permet de :

Mettre à jour la BD du fait de la modification qu'on vient d'y apporter. La mise à jour se fait après avoir enregistré les nouvelles données. Cette mise à jour concerne les attributs calculés et toutes les formes de duplication d'information en général (attributs calculés, fusion de tables, attribut créant un lien direct en plus des liens indirects).

A noter que : un trigger AFTER ne peut pas modifier de données du tuple en cours de modification. Les attributs calculés du tuple en cours de modification sont gérés par un trigger BEFORE.

6 types de triggers

Trigger before insert	Trigger after insert
Trigger before update	Trigger after update
Trigger before delete	Trigger after delete

NEW et OLD

-NEW

➡NEW est un mot-clé : il permet d'accéder aux nouvelles valeurs du tuple qu'on est en train d'ajouter ou de modifier.
➡NEW s'utilise dans les triggers INSERT et UPDATE uniquement.
En cas de DELETE, il n'y a pas de nouvelles valeurs.

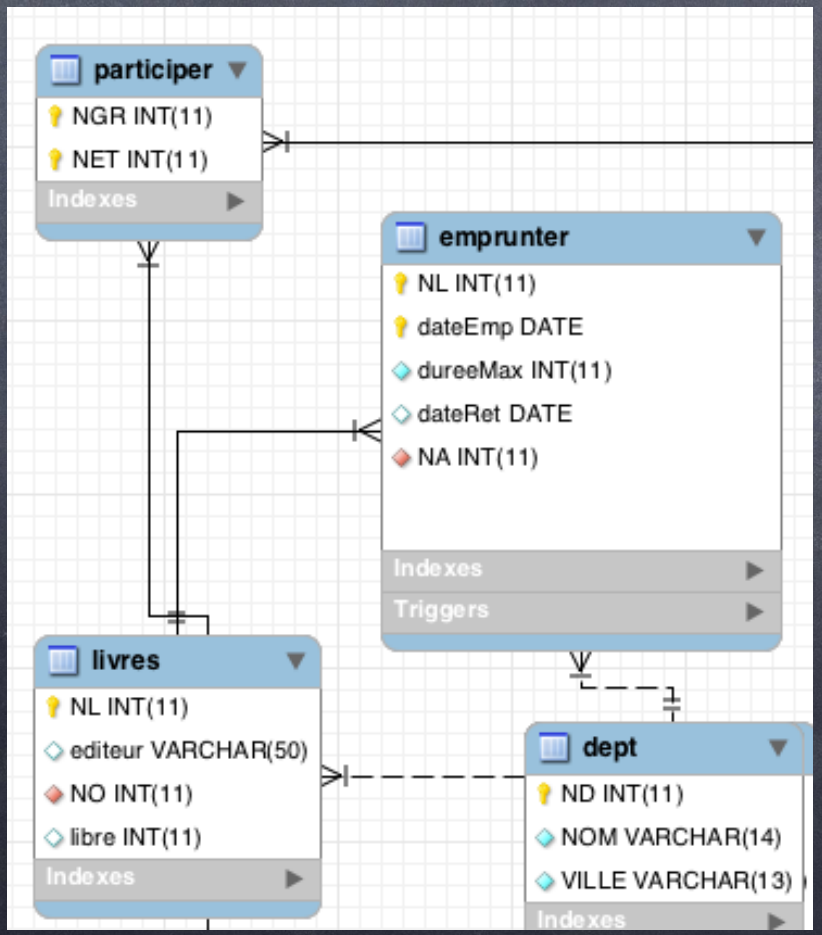
-OLD

➡OLD est un mot-clé : il permet d'accéder aux anciennes valeurs du tuple qu'on est en train de modifier ou de supprimer.
➡OLD s'utilise dans les triggers UPDATE et DELETE uniquement.
En cas d'INSERT, il n'y a pas d'anciennes valeurs.

Exemple : si on modifie Lucas en Lukas
new.prenom -> Lukas
old.prenom -> Lucas

Exemple et syntaxe

Exemple soit une BD biblio.
le champ « libre » dans la table Livres indique si le livre est disponible ou pas.



Exemple et syntaxe

Exemple soit une BD biblio.
le champ « dispo » dans la table Livres indique si le livre est disponible ou pas.

```
drop trigger if exists tai_emprunter;
delimiter //
create trigger tai_emprunter -- tai pour trigger after insert
    after insert on emprunter
    for each row
begin
    update livres set dispo=0 where nl=new.nl;
end;
//
delimiter ;
```

Après un ajout dans la table emprunter. On récupère le numéro du livre concerné (new.nl). On le cherche dans la table livres et on passe la dispo à 0. (plus disponible)

Explications

La trigger est d'abord supprimé s'il existait déjà.

Avant la création du trigger, il faut changer de délimiteur : delimiter //.

Ceci vient du fait que le code du trigger utilise des « ; » comme délimiteur d'instruction et que le « ; » est le délimiteur d'instruction standard du SQL.

La commande de création d'un trigger est : create trigger ... end ;
Entre create et begin, on définit les caractéristiques du trigger : son nom, before ou after, l'action du DML, la table concernée.

Explications

For each row veut dire qu'en cas action du DML sur plusieurs tuples, l'action du trigger se fera pour chaque tuple. MySQL ne permet pas d'autre alternative.

Le corps du trigger commence par « begin » et finit par « end ; »
Dans le corps de la procédure on peut mettre des requêtes SQL et utiliser des éléments de programmation du PL-SQL.

On termine l'instruction SQL de création du trigger par le délimiteur « // ».

On revient au délimiteur standard : « ; »

Exemple et syntaxe

Dans la BD employés, on considère qu'un salaire ne peut pas être négatif.

```
drop trigger if exists tbi_emp;
delimiter //
create trigger tbi_emp
    before insert on emp
    for each row
begin
    if new.sal  < 0 then
         -- on garde un message d'erreur dans une variable
         select 'le salaire ne peut pas être négatif'  into @tgerr;
         -- on fait planter le trigger !!!
         select 'a','a'  into @tgerr;
    end if;
end ;
//
delimiter ;
```

La solution proposée est une « bidouille » pour faire planter le trigger : l'instruction : « select 'a','a' into @tgerr; » fait planter le trigger car on ne peut pas mettre deux valeurs dans une variable.

Exemple et syntaxe

Dans la BD employés, on considère qu'un salaire ne peut pas être négatif.

```
mysql> insert into emp (nom, sal, ND) values ('DURAND', -100, 10);

ERROR 1222 (21000): The used SELECT statements have a different
    number of columns
mysql> select @tgerr;
+-------------------------------------+
| @tgerr                              |
+-------------------------------------+
| le salaire ne peut pas être négatif |
+-------------------------------------+
1 row in set (0.00 sec)

mysql> insert into emp (nom, sal, ND) values ('DURAND', 100, 10);

Query OK, 1 row affected (0.13 sec)
```

A la place de la solution précédente, qui n'est pas très élégante, le SQL-ANSI définit
l'instruction SIGNAL qui permet de gérer l'interruption et les messages.

Unicité des triggers

- On ne peut avoir qu'un et un seul trigger BEFORE par action de DML pour une table donnée.

- On ne peut avoir qu'un et un seul trigger AFTER par action de DML pour une table donnée.

Unicité des triggers

Dans les exemples précédents, on a 2 triggers BEFORE INSERT dans la table des employés : l'un pour mettre à jour le salaire total, l'autre pour vérifier que le salaire n'est pas négatif.

Il faut fusionner le code des deux triggers :

```
Alter table emp add saltot integer;

Update emp set saltot = sal + ifnull(comm,0);

drop trigger if exists tbi_emp;
delimiter //
create trigger tbi_emp
    before insert on emp
    for each row
begin
    -- on vérifie d'abord que les contraintes sont vérifiées :
    if new.sal  < 0 then
         -- on garde un message d'erreur dans une variable
         select 'le salaire ne peut pas être négatif'  into @tgerr;
         -- on fait planter le trigger !!!
         select 'a','a'  into @tgerr;
    end if;

    -- Ensuite on traite les attributs calculés du tuple
    -- sal est un attribut not NULL à la différence de comm
    set new.saltot=new.sal+ifnull(new.comm,0);
end ;
//
delimiter ;
```

SYSTEME DE TRIGGERS

En général, il ne faut pas créer un trigger unique mais un système de trigger qui prend en compte les trois possibilités du DML : INSERT, UPDATE et DELETE.

SYSTEME DE TRIGGERS : triggers BEFORE

Triggers BEFORE :

En général, le système de triggers se limite à un BEFORE INSERT et un BEFORE UPDATE.
Toutefois, il est possible de vérifier que la suppression n'entraîne pas d'incohérence et de l'empêcher sinon.

Exemple
Si on veut par exemple forcer la mise en majuscule d'un attribut, il faudra le faire en cas d'INSERT mais aussi sur les UPDATE.

SYSTEME DE TRIGGERS : triggers AFTER

Triggers AFTER :

Pour les triggers AFTER, en générale le système sera constitué d'un trigger pour chaque opération du DML : AFTER INSERT, AFTER UPDATE et AFTER DELETE.

Exemple

Dans le cas de la bibliothèque, on a fait un trigger AFTER INSERT dans la table « emprunter » pour mettre à jour l'attribut « dispo » -qui passe à faux- de la table « livres ». Il faut aussi créer un trigger AFTER UPDATE dans la table « emprunter » pour faire passer l'attribut « dispo » à « vrai ». On pourrait aussi imaginer un trigger AFTER delete.

Test

```
drop trigger if exists tbi_emp;
delimiter //
create trigger tbi_emp
before insert on emp
for each row
begin
     if new.sal  < 0 then
       -- Attention ! Pas implémenté en MySQL 5.0
       SIGNAL SQLSTATE '80000' SET MESSAGE_TEXT= 'le salaire ne
       peut pas être négatif' ;
     end if;
end ;
//
delimiter ;
```