Bases de données
PL-SQL
MySQL – Triggers

Bertrand LIAUDET

SOMMAIRE

Edition sept 2019

PL-SQL : LES TRIGGERS (DECLENCHEURS)

http://dev.mysql.com/doc/refman/8.0/en/triggers.html

http://dev.mysql.com/doc/refman/8.0/en/create-trigger.html

Présentation

Présentation

Définition

Un trigger est un programme déclenchée automatiquement quand il se passe quelque chose dans le SGBD.

Classification

On peut définir trois grandes catégories de trigger :

- Les triggers DDL
- Les triggers DML
- Les triggers de remplacement

Les triggers DDL

Les triggers DDL peuvent être déclenchés :

- sur une action du DDL : CREATE, DROP ou ALTER
- quand un utilisateur se connecte ou se déconnecte de la BD

Ces triggers ne sont pas standards. On les trouve particulièrement dans SQL Server.

Les triggers DML

Un trigger est un programme associée à une table et qui est déclenchée automatiquement avant (BEFORE) ou après (AFTER) une action du DML : INSERT, UPDATE ou DELETE d'un tuple de la table.

Les triggers de remplacement

Un trigger DML peut aussi permettre de remplacer l'instruction déclencheuse par une autre action.

Triggers DML : trigger BEFORE et trigger AFTER

Définition

Un trigger est un programme associé à une table et qui est déclenchée automatiquement avant (**BEFORE**) ou après (**AFTER**) une action du DML : **INSERT**, **UPDATE** ou **DELETE** d'un tuple de la table.

Améliorer l'intégrité

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.

Trigger BEFORE

Un trigger BEFORE est un trigger qui se déclenche avant l'action du DML (INSERT, UPDATE, DELETE). Il permet 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.

  Ce type de trigger permet par exemple de vérifier l'intégrité référentielle dans une base de données MyISAM qui ne gère pas l'intégrité référentielle.

- **Mettre à jour le tuple en cours de modification** : cela peut se faire de deux façons :
  - ✓ **Reformater les données saisies** : on peut passer du texte en majuscules.
  - ✓ **Gérer les attributs calculés du tuple en cours de modification**. Cette possibilité correspond à un trigger AFTER mais ce dernier ne peut pas s'appliquer au tuple en cours de modification.

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.

Les 6 types de triggers : BEFORE / AFTER croisés à INSERT / UPDATE / DELETE

On distinguera entre 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 trigger 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 trigger UPDATE et DELETE uniquement. En cas d'INSERT, il n'y a pas d'anciennes valeurs.

Exemples et syntaxe

TRIGGER AFTER

Dans la BD biblio, on ajoute le champ « dispo » dans la table LIVRES : c'est un champ calculé qui dit 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 ;
```

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.
- 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. ORACLE permet de créer des triggers qui s'appliquent après la modification de tous les tuples.
- 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 : « ; »

TRIGGER BEFORE

On va présenter un exemple pour deux des trois usages du trigger BEFORE.

Gérer les attributs calculés du tuple en cours de modification

Dans la BD employés, on ajoute le champ « saltot » dans la table EMPLOYES : c'est un champ calculé qui calcule le salaire total (sal + comm).

On met à jour la valeur de cet attribut pour tous les tuples.

Enfin, on crée le trigger.

```
Alter table employes add saltot integer;

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

drop trigger if exists tbi_employes;
delimiter //
create trigger tbi_employes
    before insert on employes
    for each row
begin
    -- sal est un attribut not NULL à la différence de comm
    set new.saltot=new.sal+ifnull(new.comm,0);
end;
//

delimiter ;
```

Interdire l'action du DML en cas d'incohérence des données saisies

- ***Exemple***

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

```
drop trigger if exists tbi_employes;
delimiter //
create trigger tbi_employes
    before insert on employes
    for each row
begin
    if new.sal  < 0 then
         SIGNAL SQLSTATE '80001' SET MESSAGE_TEXT=
         'le salaire ne peut pas être négatif';
    end if;
end;
//

delimiter ;
```

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

ERROR 1644 (80001): le salaire ne peut pas être négatif
```

- ***l'instruction SIGNAL (depuis la version 5.5)***

Depuis la version 5.5, MySQL définit l'instruction SIGNAL qui permet de gérer l'interruption et les messages.

L'instruction SIGNAL permet de gérer des arrêts brutaux et aussi de simples warnings. Pour le détail, se reporter à la doc MySQL :

http://dev.mysql.com/doc/refman/8.0/en/signal.html

➢ ***Solution alternative pour les versions antérieures***

```
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 ;
```

```
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)
```

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.

Syntaxe

CREATE [DEFINER={*user* | CURRENT_USER}] TRIGGER *nomTrigger*

{ BEFORE | AFTER } { UPDATE | INSERT | DELETE } ON *tableName*

FOR EACH ROW

InstructionsDuTrigger

Rappel de la sémantique du métalangage:

Majuscule : les mots-clés

Entre crochets : ce qui est facultatif

Entre accolades : proposition de plusieurs choix possibles. Les choix sont séparés par des barres verticales.

En minuscule et italique : le nom d'une variable ou d'une série d'instructions.

Précisions

[DEFINER={*user* | CURRENT_USER}] : permet de limiter l'usage du trigger à un utilisateur.

nomTrigger : c'est le nom donné à la procedure trigger.

{ BEFORE | AFTER } : choix du déclenchement avant ou après l'opération du DML.

{ UPDATE | INSERT | DELETE }: choix de l'opération du DML.

tableName : nom de la table à laquelle est associé le trigger.

donnée à la procedure trigger.

FOR EACH ROW : le trigger s'applique à tous les tuples affectés par les traitements. MySQL ne permet que cette option.

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.

Exemple

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 :

```
-- drop column SALTOT si on a déjà créé SALTOT
Alter table EMPLOYES drop column SALTOT;
Alter table EMPLOYES add SALTOT integer;

Update EMPLOYES
set SALTOT = SAL + ifnull(COMM,0);

drop trigger if exists tbi_employes;
delimiter //
create trigger tbi_employes
   before insert on EMPLOYES
   for each row
begin
   -- on vérifie d'abord que les contraintes sont vérifiées :
   if new.SAL  < 0 then
        SIGNAL SQLSTATE '80001' SET MESSAGE_TEXT=
        'le salaire ne peut pas être négatif';
   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 ;
```

Pour créer ce code sous php-myadmin, il faut traiter séparément le SQL (alter, update, drop) et le PL-SQL (create trigger).

Pour le create trigger, il ne faut pas gérer le delimiter dans le code, mais dans la fenêtre php-myadmin : on met le « // ».

On peut aussi utiliser l'interface graphique de php-myadmin pour créer des déclencheurs.

Système de triggers

Principe général

En général, il faut 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.

Cas des triggers BEFORE

Principe

Les systèmes de triggers BEFORE sont liés à l'usage du trigger.

- **Trigger BEFORE pour interdire l'action du DML en cas d'incohérence des données saisies :**
  - En d'INSERT ou d'UPDATE sur une table jointe, il faut vérifier la cohérence dans la table de référence.
  - En cas d'UPDATE ou de DELETE sur une table de référence, il faut vérifier la cohérence dans la ou les tables jointes.
- **Trigger BEFORE pour mettre à jour le tuple en cours de modification.**
  - Le DELETE n'est pas concerné. INSERT et UPDATE seront concernés.

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.

Cas des triggers AFTER

Principe

Les triggers AFTER concernent la mise à jour de données calculées.

Souvent, ils concernent deux tables : celle avec l'attribut calculé et celle dont dépend l'attribut calculé. En général le système sera constitué d'un système de triggers AFTER (INSERT, UPDATE et DELETE) pour la table dont dépend l'attribut calculé, et d'un système de triggers BEFORE (INSERT et UPDATE) pour la table de l'attribut calculé.

Exemple

Soit le MR suivant :

- EMP(NE, nom, #ND)
- DEPT(ND, nom, **nbEmp**)

Les employés travaillent dans un département. Dans le département, on trouve **l'attribut calculé « nbEmp »**.

Les triggers after

- EMP(NE, nom, #ND)
- DEPT(ND, nom, **nbEmp**)

A insert EMP	Update Dept set nbEmp++ where ND=new.ND
A delete EMP	Update Dept set nbEmp - - where ND=old.ND
A update EMP	Update Dept set nbEmp - - where ND=old.ND Update Dept set nbEmp++ where ND=new.ND Si on fait un update, peut-être que ND change : dans ce cas, on décrémente le old.ND puis on incrémente le new.ND. Si on a changé de ND, les mises à jour sont faites. Si on n'a pas changé de ND, on revient à la situation intiale.

A insert DEPT	-
A update DEPT	-
A delete DEPT	-

Les triggers before

- EMP(NE, nom, #ND)
- DEPT(ND, nom, **nbEmp**)

B insert DEPT	Set nbEmp=0
B update DEPT	Ici on veut interdire l'update manuel car l'attribut n'est pas modifiable. On peut coder : If old.nbEmp != new.nbEmp then set new.nbEmp=old.nbEmp ATTENTION : A NE PAS FAIRE !!! En effet bloquer l'update manuel va aussi annuler l'update automatique qui vient du trigger after de EMP qui justement modifie nbEmp (++ ou --) Mais on peut faire : Select count(*) from EMP Where nd=new.nd into nb; set new.nbEmp=nb; A noter que du coup, l'update effectué dans les triggers after update, inserte ou delete de EMP n'a plus aucune importance ! On voit à cette occasion que la gestion des attributs calculés par triggers est un choix très discutable d'autant que le count(*) peut être couteux. Il faudrait au moins indexer #nd dans emp pour optimiser le count.
B delete DEPT	-

B insert EMP	-
B update EMP	-
B delete EMP	-

Code du trigger before update on departement

```
drop trigger if exists tbu_dept;
delimiter //
create trigger tbu_dept
    before update on departements
    for each row
begin
    declare nb int;
    select count(*) from employes Where nd=new.nd into nb;
    set new.nbEmp=nb;
end ;
//
delimiter ;
```

Conclusion : attention aux attributs calculés !!!

Pour être complets, les triggers AFTER doivent être constitués en système cohérent. Mais il faut faire attention ! Il faut les manipuler avec précaution ! De plus, il y a un risque de dégradation des performances puisqu'on voit dans l'exemple qu'il faut faire un count(*) ce qui peut être couteux.

Utilisation d'une vue plutôt que d'un attribut calculé

Puisque l'objectif concerne la mise à jour d'un attribut calculé, le mieux est d'utiliser une vue :

```
Create or replace view departements2 as
select d.nd, d.nom, d.ville, count(*) nbEmp
from employes e, departements d
where e.nd=d.nd
group by d.nd, d.nom, d.ville;
```

```
show tables;
select * from departements2;
```

Toutefois cette vue ne liste pas les départements avec 0 employés.

On peut résoudre le problème avec la requête suivante :

```
Create or replace view departements2 as
select d.nd, d.nom, d.ville, count(*) nbEmp
from employes e, departements d
where e.nd=d.nd
group by d.nd, d.nom, d.ville
union
select nd, nom, ville, 0 from dept
where nd not in (select nd from emp);
```

```
show tables;
select * from departements2;
```

Ou encore avec une jointure externe à la place du not in

```
Create or replace view departements2 as
select d.nd, d.nom, d.ville, count(*) nbEmp
from employes e, departements d
where e.nd=d.nd
group by d.nd, d.nom, d.ville
union
select d.nd, d.nom, d.ville, 0
from employes e right join departements d
using (nd)
where e.ne is null;
```

```
show tables;
select * from departements2;
```

Méthode de nommage des vues

On peut imaginer avoir un nom de base pour toutes les tables de la BD, par exemple tout préfixer par « sys_ » : sys_employes, sys_departements.

Ensuite on donne accès aux vues que l'on souhaite : departement avec l'attribut calculé « nbEmp », etc.

Gestion des triggers

Afficher les triggers existants :

```
Show triggers ;
```

Afficher le code d'un trigger:

```
Show triggers ;
```

Usage des triggers

Les triggers sont déclenchés automatiquement quand l'événement DML déclencheur arrive :

```
mysql> insert into emp (NE, nom, sal, comm, ND) values (999, 'TOTO', 1200, 150, 10);

mysql> select * from emp where NE=999;
+------+--------+-----------+------------+---------+---------+----+-------+---------+
| NE   | NOM    | JOB       | DATEMB     | SAL     | COMM    | ND |NEchef | saltot |
+------+--------+-----------+------------+---------+---------+----+-------+---------+
| 99 9 | TOTO   | NULL      | NULL       | 1200.00 |  150.00 | 10 |  7839 | 1200.00|
+------+--------+-----------+------------+---------+---------+----+-------+---------+
1 row in set (0.01 sec)
```

Débogage

On ne peut pas afficher des commentaires dans un trigger via un select.

En cas d'erreur de compilation, il faut se référer au message d'erreur envoyé par le serveur.

En cas d'erreur d'exécution, on peut utiliser des variables globales et faire un select dans une variable globale.

Afficher les erreurs-warnings :

```
Show warnings ;
```

Affichage des triggers avec le dictionnaire : BD information_schema

```
mysql> select trigger_name from information_schema.triggers;
+--------------+
| trigger_name |
+--------------+
| tbi_emp      |
+--------------+
1 row in set (0.05 sec)
```

```
mysql> select action_statement from information_schema.triggers;
+-----------------------------------------------------------------
| action_statement
+-----------------------------------------------------------------
| begin
        if new.sal  < 0 then

                SIGNAL SQLSTATE '80001' SET MESSAGE_TEXT=
                'le salaire ne peut pas être négatif' ;
        end if ;
end |
+-----------------------------------------------------------------
1 row in set (0.05 sec)
```

```
mysql> select * from information_schema.triggers\G;
*************************** 1. row ***************************
           TRIGGER_CATALOG: def
            TRIGGER_SCHEMA: empdept
              TRIGGER_NAME: tbi_emp
        EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: def
       EVENT_OBJECT_SCHEMA: empdept
        EVENT_OBJECT_TABLE: emp
              ACTION_ORDER: 0
          ACTION_CONDITION: NULL
          ACTION_STATEMENT: begin
        if new.sal  < 0 then

                SIGNAL SQLSTATE '80001' SET MESSAGE_TEXT=
                'le salaire ne peut pas être négatif' ;
        end if ;
end
        ACTION_ORIENTATION: ROW
             ACTION_TIMING: BEFORE
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
  ACTION_REFERENCE_OLD_ROW: OLD
  ACTION_REFERENCE_NEW_ROW: NEW
                  CREATED: NULL
                SQL_MODE:
                 DEFINER: root@localhost
      CHARACTER_SET_CLIENT: cp850
      COLLATION_CONNECTION: cp850_general_ci
        DATABASE_COLLATION: latin1_swedish_ci
1 row in set (0.06 sec)
```

Affichage des triggers via une interface graphique

MySQL WorkBench

PhpMyAdmin

- ***Onglet plus/declencheur***

On précise le nom du trigger, la table, before ou after, insert update ou delete.

Puis le « corps » du trigger : la partie entre Begin de début et End final.

Le créateur est par exemple : root@localhost, ce qui veut dire l'utilisateur root sur la machine du serveur (localhost). On peut aussi ne rien mettre.

Le trigger apparaît dans le navigateur du SGBD dans le dossier de la table auquel il correspond.

Eléments de PL-SQL

On présente ici quelques éléments de PL-SQL utile pour l'écriture des triggers.

Variable locale

Principe

Les variables locales sont déclarées dans le corps du trigger. Elles ne sont visibles que dans le trigger.

Déclaration : DECLARE

```
declare my_int int;
declare my_num numeric(8,2);
declare my_pi float default 3.1415926;
declare my_text text;
declare my_date date default '2008-02-01';
declare my_varchar varchar(30) default 'bonjour';
```

Affectation 1 : SET

```
set my_int=20;
set my_bigint = power(my_int,3);
set my_date = current_date;
```

Affectation 2 : SELECT ... INTO

```
Select 20 into my_int;
Select nom into my_varchar from emp limit 1;
```

Affichage d'une variable locale dans un programme

```
select my_varchar, my_int;
```

Remarque : on ne peut rien afficher dans un trigger !

Variables globales : @

Déclaration et affectation : @ et SET

```
set @gbint=99;
set my_int=@gbint;
```

Déclaration et affectation : @ et SELECT ... INTO

```
Select nom into @gbchar from emp limit 1;
```

Affichage d'une variable globale dans un programme

```
Select @gbchar;
```

Affichage d'une variable globale dans le SGBD

```
mysql > Select @gbchar;
```

Commentaires

```
/*   script de définition d'une procédure
     procédure « bonjour » : affiche bonjour,
     usage des commentaires en style C
*/
-- commentaires derrière deux tirets et un espace
```

Opérateurs et fonctions accessibles

http://dev.mysql.com/doc/refman/5.6/en/functions.html

Opérateurs de comparaison :

>, <, <=, >=, BETWEEN, NOT BETWEEN, IN, NOT IN, =, <>, !=, like, regexp (like étendu), isnull, is not null, <=>.

Opérateurs mathématiques :

+, -, *, /, DIV, %

Opérateurs logiques :

AND, OR, XOR

Opérateurs de traitement de bit : | , &, <<, >>, ~

Fonctions de contrôle :

ifnull, if, case, etc.

Fonctions de chaîne de caractères :

substring, length, concat, lower, upper, etc.

Fonctions numériques :

abs, power, sqrt, ceiling, greatest, mod, rand, etc.

Fonctions de dates et d'heures :

current_date, current_time, to_days, from_days, date_sub, etc.

Fonctions de recherche en texte intégral :

match

Fonctions de transtypage :

cast et convert

Autres fonctions

Tests – IF – ELSE - ELSIF

Exemple

```
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 ;
```

Syntaxe

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

- ***Rappel de la sémantique du métalangage:***

Majuscule : les mots-clés

Entre crochets : ce qui est facultatif

Entre accolades : proposition de plusieurs choix possibles. Les choix sont séparés par des barres verticales.

En minuscule et italique : le nom d'une valeur, d'une variable, d'une expression, d'une instruction ou d'une série d'instructions.

CASE WHEN

Syntaxe

```
CASE expression
WHEN valeurs THEN
    instructions
[WHEN valeurs THEN
    instructions ]
[ELSE
    instructions ]
```

Rappels de DDL : ALTER TABLE

http://dev.mysql.com/doc/refman/5.6/en/alter-table.html

Modification des attributs

Ajouter un ou plusieurs attributs à la table :

```
ALTER TABLE NomTable  ADD (
        attribut_1  type [contrainte],
        attribut_2  type [contrainte],
        … ,
        attribut_n  type [contrainte]
);
```

Modifier un attribut de la table

```
ALTER TABLE NomTable  MODIFY
        attribut_1  type [contrainte]
;
```

La modification permet d'annuler les contraintes de type NOT NULL ou auto_increment.

Supprimer un attribut de la table

```
ALTER TABLE NomTable DROP attribut ;
```

- ***Attention***

La modification et la suppression des attributs doivent être manipulées avec prudence : une table peut contenir des milliers de données. Il ne faut pas les supprimer ou modifier une table sans précaution.

Ajout et suppression de contraintes d'intégrité

Ajouter une contrainte

```
ALTER TABLE NomTable  ADD [ CONSTRAINT  nomContrainte]
Contrainte ;
```

- ***Exemple***

```
ALTER TABLE emp
ADD CONSTRAINT  keynd FOREIGN KEY(ND) REFERENCES DEPT(ND);
```

```
ALTER TABLE emp  ADD foreign key(ND) references DEPT(ND);
```

Suppression d'une contrainte nommée

```
ALTER TABLE NomTable  DROP type de contrainte
nom_de_contrainte;
```

- ***Exemple***

```
ALTER TABLE emp  DROP foreign key  KEYND;
```

Récupérer le nom des contraintes : show create table *maTable*

Si on n'a pas nommé les contraintes à la création, MySQL les nomme automatiquement. Pour récupérer le nom, il faut utiliser la commande

```
Show create table maTable
```

On obtient alors le nom de la contrainte derrière le mot clé CONSTRAINT.

Suppression de la clé primaire

```
ALTER TABLE NomTable  DROP primary key ;
```

On ne peut supprimer la clé primaire que si ce n'est pas un auto incrément, et uniquement si elle n'est pas référencée par une clé étrangère.

TP TRIGGERS

0. MySQL Workbench

Pour écrire les procédures stockées et triggers, on peut utiliser le MySQL Workbench.

Présentation

MySQL Workbench fournit aux DBAs et aux développeurs un environnement de développement intégré pour :

- La conception de la BD
- Le développement SQL (MySQL Workbench remplace MySQL Tools : MySQL administrator et MySQL Query Browser)
- L'administration de la BD

Téléchargement

http://www.mysql.fr/downloads/workbench/

Télécharger : Windows (x86, 32-bit), MSI Installer

Installation

Pas de paramétrage : version complète.

Résultats de l'installation

Répertoire d'installation

C:\Program Files\MySQL\MySQL Workbench 5.2 CE

Fichier exécutable

C:\Program Files\MySQL\MySQL Workbench 5.2 CE\MySQLWorkbench.exe

Principes du fonctionnement

3 usages

- SQL developement
- Data Modeling
- Server Administration

SQL developement

Si le server MySQL est lancé, quand on clique sur le « local instance MySQL » ça ouvre une fenêtre qui donne accès à un « browser » avec les BD déjà enregistrées sur le serveur et à une fenêtre qui permet de passer des commandes SQL.

SQL Administration

Si le server MySQL est lancé, quand on clique sur le « local MySQL » ou sur « Server Administration, ça ouvre une fenêtre qui permet de faire de l'administration : suivre l'état du serveur, gérer les paramètres de configuration, gérer les utilisateurs, gérer les import-export de BD.

Data Modeling

- Create new EER model / add diagram : on peut créer des tables et des liaisons et enregistrer le modèle
- Open existing EER model : permet d'ouvrir un modèle précédemment enregistré
- Create EER model from existing database : permet de créer un modèle à partir d'une BD déjà enregistrée sur le serveur
- Create EER model from SQL script : permet de créer un modèle à partir d'un fichier script SQL.

1. TD : les employés

Vérifier que le salaire est positif

1. A partir de la base « empdept », coder le trigger permettant de vérifier que le salaire saisi est positif. Vous pouvez prendre le code du cours : il génère une erreur : lisez bien le message affiché pour essayer de trouver l'erreur.
2. Une fois l'erreur corrigée, affichez la liste des triggers enregistrés dans votre BD.
3. Affichez le code du trigger que vous venez de créer.

Coder le système de triggers de l'attribut calculé « nbEmp »

4. A partir de la base « empdept » contenant une table d'employés et une table de département, le principe étant qu'un employé travaille obligatoirement dans un département et un seul, **concevoir** le système de triggers permettant de gérer un attribut calculé donnant dans la table des départements le nombre d'employés : cf. cours. Quels triggers sont nécessaires. Quels triggers sont à éviter !
5. Coder le bon système.
6. Affichez la liste des triggers enregistrés dans votre BD.

2. La bibliothèque - triggers

Créer la BD « biblio » à partir du script fourni.

1. On souhaite ajouter l'attribut « emprunté » qui est un booléen et précise si un livre est actuellement emprunté ou pas. Ecrire la requête qui permet d'ajouter cet attribut (ALTER TABLE).
2. Mettre à jour les valeurs de cet attribut pour tous les tuples de la table concernée (UPDATE).
3. Concevoir puis coder le système de triggers qui permet de gérer cet attribut calculé.
4. On souhaite ajouter l'attribut « dureeEmprunt » qui donne la durée de l'emprunt en jours après le retour du livre. Ecrire la requête qui permet d'ajouter cet attribut (ALTER TABLE).
5. Mettre à jour les valeurs de cet attribut pour tous les tuples de la table concernée (UPDATE).
6. Ecrire le système de triggers qui permet de gérer cet attribut calculé.
7. On souhaite que la date d'emprunt soit désormais un attribut automatique qui vaut automatiquement la date du jour de la création du tuple correspondant dans la BD. Comment faire ça ? Ecrire le code correspondant.
8. On souhaite que les noms de famille soient en majuscules et les prénoms en minuscules avec seulement la première lettre en majuscule. Ecrire le système de triggers qui permet de gérer cette demande.
9. On souhaite interdire l'emprunt d'un livre déjà emprunté. Ecrire le trigger.
10. On souhaite interdire l'emprunt de plus de 3 livres par le même adhérent. Ecrire le trigger.
11. On souhaite que la date de retour prenne automatiquement la valeur de la date du jour du rendu du livre. On souhaite interdire un rendu le jour même de l'emprunt et que la date de retour soit postérieure à la date d'emprunt. Ecrire le trigger.

3. Les ventes de produits

Créer la BD des produits à partir du script fourni.

1. Actuellement, on gère une date de commande et une date de livraison (date d'envoi de la commande) dans la table commande.
2. On souhaite faire évoluer le système en permettant une livraison par morceau. On va donc gérer une date d'envoi au niveau de chaque détail de commande. La date d'envoi de la commande correspond finalement à la date d'envoi du dernier détail de commande.
3. Mettre à jour la BD en conséquence.
4. Créer un système de triggers qui permette de mettre à jour automatiquement la date d'envoi au niveau de la commande.
5. Créer un système de trigger qui vérifie que la date d'envoi au niveau de la commande est cohérente, c'est-à-dire qu'elle n'existe que si tous les détails de commandes sont envoyés et qu'elle correspond à la date d'envoi la plus récente des détails de commande.