Les déclencheurs (Triggers)

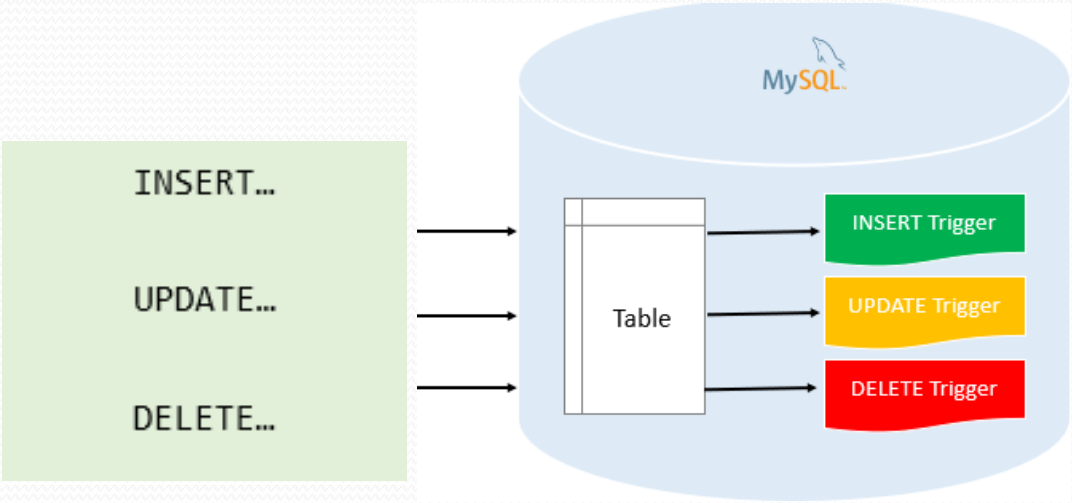
Définition de déclencheur(trigger)

- Un triggers (ou déclencheur) est un programme SQL stocké sous forme **d'objet attaché à une table** de la base de données**.**

- Les triggers vont **déclencher** l'exécution d'une instruction, ou d'un bloc d'instructions, lorsque il y'a un **événement** qui arrive au niveau de la **table à laquelle est attaché le trigger.**

Les événements déclenchant un trigger

- Les **événements** qui peuvent **déclencher un trigger** sont les suivants:
  1. Une insertion dans la table (requête **INSERT**) ;
  2. La suppression d'une partie des données de la table (requête **DELETE**) ;
  3. La modification d'une partie des données de la table (requête **UPDATE**).



Le rôle des triggers

- Les déclencheurs sont utiles dans les cas suivants :

  - Il peut permettre de **vérifier les données au moment** de la **modifications des données de la table à laquelle il est attaché,** comme la vérifications des contraintes foreign key par exemple, ce qui est fait automatiquement par le SGBD

  - Pour assurer l'application de restrictions plus complexes que celles définies à l'aide de contraintes CHECK. Contrairement aux contraintes CHECK, ils peuvent faire référence à des colonnes d'autres tables.

  - Pour détecter la différence entre l'état d'une table avant et après une modification des données, et entreprendre une ou plusieurs actions en fonction de cette différence.

Exécution d'un trigger

- Pour exécuter une procédure stockée, on fait appel avec la commande **call,** et pour exécuter une fonction on fait appel à l'aide d'un **select**

- Pour exécuter un trigger, on fait pas un appel explicite , mais on peut juste le déclencher **par l'exécution de la commande de mise à jour** adéquate : (insert into table values(), ou update table set .....ou delete * from .....).

- Un trigger peut exécuter ses instructions soit **avant ou après** son événement déclencheur.

Création d'un trigger

```
CREATE  TRIGGER  NOM_TRIGGER  MOMENT_TRIGGER
        EVENEMENT_TRIGGER  ON  NOM_TABLE  FOR EACH ROW
Begin
CORPS_TRIGGER
End
```

- **MOMENT_TRIGGER** : c'est le moment d'exécution du trigger (**avant** ou **après**)
- **EVENEMENT_TRIGGER** : Evénement déclenchant le trigger(**INSERT**, **UPDATE**, **DELETE**)
- **ON NOM_TABLE** : c'est là qu'on définit à quelle table le trigger est attaché.
- **FOR EACH ROW** : signifie littéralement "**pour chaque ligne**"

CORPS_TRIGGER : ce sont les instructions exécutées par le trigger.

Le moment d'exécution d'un trigger

- Une fois le trigger déclenché, ses instructions peuvent être exécutées soit **juste avant** l'arrivé de l'événement déclencheur, soit **juste après (BEFORE ou AFTER)**.

- Si vous avez un trigger **BEFORE UPDATE** sur la table A, l'exécution d'une requête UPDATE sur cette table va **d'abord** déclencher l'exécution des instructions du trigger, ensuite des lignes de la table seront modifiées.

- Si vous avez un trigger **AFTER UPDATE** sur une table A, l'exécution d'une requête UPDATE sur cette table va modifier les lignes concernées puis l'exécution des instructions du trigger.

Les lignes affectés (FOR EACH ROW)

- Un trigger exécute un traitement **pour chaque ligne (each row)** insérée, modifiée ou supprimée par l'événement déclencheur.
- Par exemple si on insère cinq lignes à la fois, les instructions du trigger seront exécutées cinq fois, chaque itération permettant de traiter les données d'une des lignes insérées.

Moment d'exécution du trigger: Avant ou après

- Exemple: Trigger qui va être exécuté avant l'insertion dans la table Commande.

```
CREATE TRIGGER CommandeValide BEFORE INSERT ON Commande FOR EACH ROW

Begin

Corps du trigger

End
```

Les mots OLD et NEW

- Dans le corps du trigger, MySQL met à notre disposition deux mots-clés : **OLD** et **NEW**.

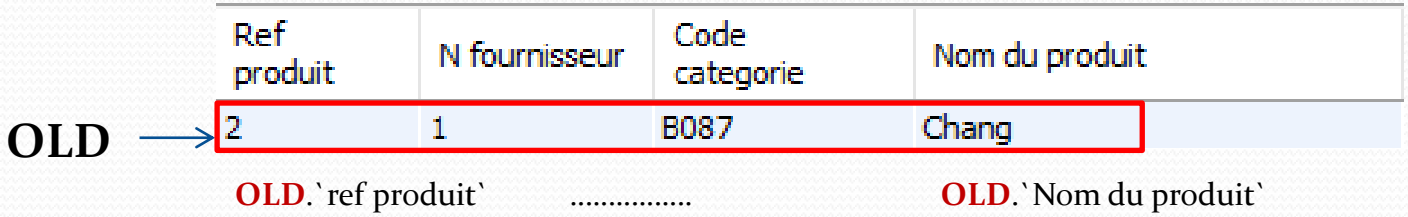
1. **OLD** : représente les valeurs des colonnes de la **ligne** traitée **avant qu'elle ne soit modifiée** par l'événement déclencheur. Ces valeurs peuvent être **lues, mais pas modifiées**.

2. **NEW** : représente les valeurs des colonnes de la ligne traitée **après qu'elle a été modifiée** par l'événement déclencheur. Ces valeurs peuvent être **lues et modifiées**.

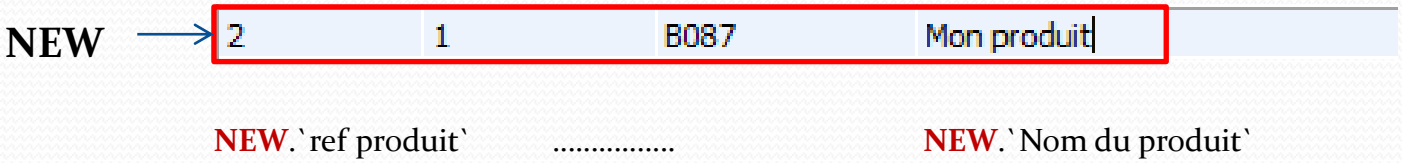
- Pour les utiliser, il suffit d'écrire: **NEW**.colonnex et **OLD**.colonnex

OLD et NEW

- Par exemple: on veut faire une modification du nom du produit numéro 2 dans la table produit.



La modification va être comme suit:



OLD et NEW

- Lors d'une insertion, **OLD** n'existe pas, puisque la ligne n'existe pas avant l'événement INSERT.
- dans le cas d'une suppression, c'est **NEW** qui n'existe pas, puisque la ligne n'existera plus après l'événement DELETE.;
- Il n'y a que dans le cas d'un trigger UPDATE que OLD et NEW coexistent.

	OLD	NEW
INSERT	non	oui
DELETE	oui	non
UPDATE	oui	oui

Exemple1

- Trigger qui vérifie le code d'une catégorie avant l'insertion d'une nouvelle catégorie.

```
Delimiter |
CREATE   TRIGGER  validerCat BEFORE  INSERT  ON  categories for each row
Begin

If   NEW.`code de categorie`  NOT   regexp '^[A-Z][0-9]{3}$'    then
SET NEW.`code de categorie`='Aooo';
End if;

End|
```

Exemple 2

- Le déclencheur suivant va permettre d'assurer que le nom soit unique lors de l'insertion d'un nouveau stagiaire:

```
CREATE  TRIGGER  validerStgaiaire  BEFORE  INSERT  ON  Stagiaires FOR
EACH ROW
Begin

If  NEW.nom  IN (select nom from Stagiaires)   then
Signal SQLSTATE '45000' SET Message_text='ce nom existe déjà dans la table';
End if;
End
```

- La valeur '45000' signifie une exception définie par l'utilisateur non gérée.
- Par exemple l'erreur 1064: signifie erreur de syntaxe.

Exemple 2(suite)

- Pour exécuter le déclencheur **validerStgaiaire**, il faudra exécuter une commande d'insertion dans la table stagiaires:

```
INSERT INTO STAGIAIRES values (10,'AISSAOUI','LAILA','F','2000-12-23',2)
```

- Si le nom AISSAOUI existe dans la table, cette ligne ne sera pas insérée.

Exemple 3

- Après insertion d'une nouvelle commande, une nouvelle ligne dans détails commande doit être insérée:

```
delimiter |
create trigger tr2 AFTER insert on commandes for each row
begin

insert into detailscommandes values(NEW.`n commande`,1,NULL,NULL,0);

end |
delimiter;
```

Exemple 3

- Pour exécuter le déclencheur 2 il faut exécuter une commande d'insertion dans la table commande.

```
INSERT INTO COMMANDES(….) VALUES
(valeur1,valeur2,…);
```

Exemple 4

```
create trigger  verifier  BEFORE UPDATE  on Stagiaires for each row
begin

If NEW.NumStagiaire!=OLD.NumStagiaire then

Signal SQLSTATE '45000' SET Message_text ='vous n avez pas le droit de
modifier le numéro de stagiaire';

end
```

Exemple 4

- Pour exécuter ce trigger il faut exécuter une commande de modification dans la table stagiaire:

```
UPDATE STAGIAIRE set NUMSTAGIAIRE=19 where
nomstagiaire='Mamoun', prenomStagiaire='ALI';
```

Remarques importantes

- Lorsque on a un trigger sur une table, on peut pas écrire dans les instructions des commandes de mise à jour de la **même table**.
- On ne peut pas créer une table l'intérieur d'un trigger.
- On ne peut pas faire une **sélection pour afficher** les données d'une table.

Combien de triggers par table?

- Le nombre maximum de triggers par table est 6. car il y'a 3 événements et 2 moments:

  - BEFOR INSERT
  - AFTER INSERT
  - BEFORE UPDATE
  - AFTER UPDATE
  - BEFORE DELETE
  - AFTER DELETE

6 triggers possibles

Supprimer un trigger

- La suppression d'un trigger est faite avec la commande DROP:

```
DROP   TRIGGER  IF EXISTS NomTrigger;
```