UNIVERSITÉ TOULOUSE Jean Jaurès

Département de Mathématiques et Informatique

MI0A301T : BD Avancées

1 - Rappels sur les SGBD Relationnels

2 – Vues relationnelles

3 – Fonctions et procédures stockées

4 – Triggers – Cours magistral

Contraintes d'intégrité complexes

- Parmi les contraintes sur les données, qu'un SGBDR peut vérifier, il y a :
  - Le typage, la taille et le caractère obligatoire des données
  - Les contraintes de clé primaire et de clé étrangère
  - Les expressions booléennes d'une clause CHECK
- La clause SQL CHECK ne permet pas de mettre en place des contraintes d'intégrité complexes.
  - Exemples de contraintes complexes :
    - Le salaire d'un enseignant ne peut pas être diminué
    - Le nombre d'étudiants en L1 MIASHS (information non stockée) est limité à 180
    - Une date de naissance doit être antérieure à la date du jour (date du système qui change tous les jours …)
- Pour cela nous pouvons utiliser des **triggers** !

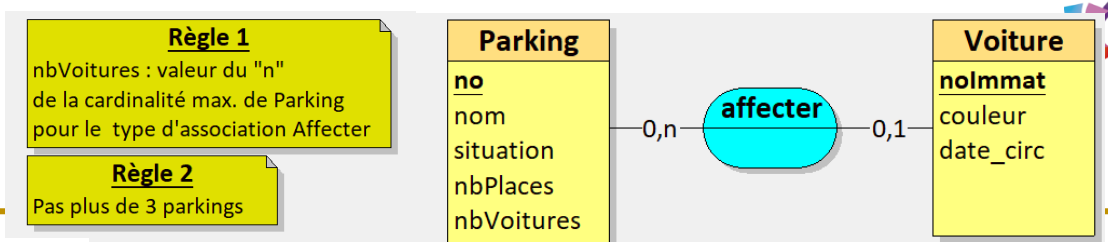
Les triggers SQL

- Un « trigger », en français « un déclencheur », est un traitement (série d'instructions) qui est **exécuté automatiquement lorsqu'un événement se produit**
- Dans un SGBD, un **trigger SQL est associé à une requête de modification d'une table** d'une base de données (requête *INSERT INTO*, *UPDATE* ou *DELETE FROM*)
- **Il se déclenche avant** (BEFORE) **ou après** (AFTER) **la réalisation de la requête de mise à jour** pour laquelle il a été défini

info Il n'existe pas de trigger déclenché par une instruction SELECT (LID)

info Certains SGBD permettent d'associer des triggers à des requêtes du LDD ; ce n'est pas le cas de MySQL/MariaDB

Exemple



- Soit une résidence avec 3 parkings (Cf. schéma E-A) :
  - Des voitures sont affectées aux parkings ; pour accélérer les traitements, on a « dénormalisé » le schéma, et on mémorise combien de voitures sont affectées à chaque parking (propriété calculable **nbVoitures**).
- La base est donc composée des relations Parkings et Voitures :
  - **Parkings (**no, nom, situation, nbPlaces, nbVoitures**)**
  - **Voitures (**noImmat, couleur, date_circ, parking**#)**
- Créées avec les requêtes SQL suivantes :

```
CREATE TABLE Parkings  (
 no INT AUTO_INCREMENT PRIMARY KEY,
 nom VARCHAR(15),
 situation VARCHAR(20),
 nbPlaces INT NOT NULL,
 nbVoitures INT DEFAULT 0 );
```

```
CREATE TABLE Voitures  (
    noImmat VARCHAR(10) PRIMARY KEY,
    couleur VARCHAR(15),
    date_circ DATE,
    parking INT,
    FOREIGN KEY (parking) REFERENCES Parkings (no) );
```

Nous allons mettre en place des triggers pour **assurer** l'intégrité de la base vis-à-vis des deux contraintes exprimées (règles 1 et 2)

Structure d'un trigger SQL

- On peut remarquer 3 parties dans un trigger :
  - Moment où le trigger est exécuté (AFTER/BEFORE)
  - Evènement déclencheur et relation concernée
  - Traitement : action(s) à réaliser lors de son exécution.

- Les 3 parties d'un trigger :

```
{AFTER | BEFORE}
{DELETE | INSERT | UPDATE}
ON nom_relation
FOR EACH ROW
BEGIN
    Instructions à réaliser ;
END
```

Moment de l'exécution

Evénement déclencheur

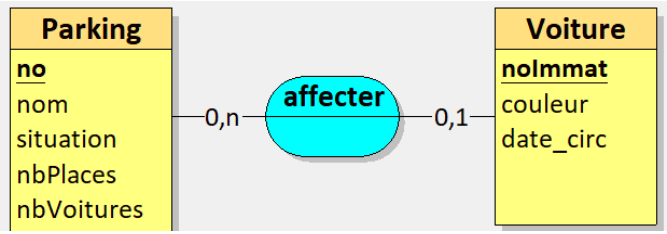
Relation concernée

Action(s) du trigger

Les triggers pour : Cas 1

- **Calculer et mémoriser des données calculables …**

- Exemple :

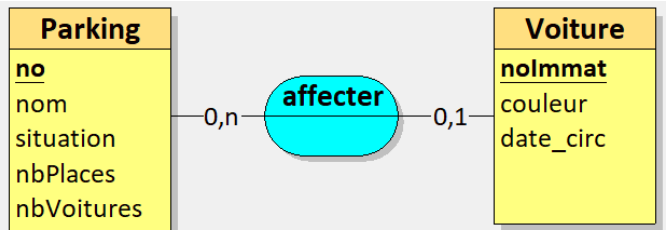


Règle 1
nbVoitures : valeur du "n"
de la cardinalité max. de Parking
pour le type d'association Affecter

  - nbVoitures mémorise au niveau d'un parking, combien de voitures lui sont affectées

Parkings (no, nom, situation, nbPlaces, nbVoitures)
Voitures (noImmat, couleur, date_circ, parking#)

Exemple Cas 1



Parkings (no, nom, situation, nbPlaces, nbVoitures**)**
Voitures (noImmat, couleur, date_circ, parking#**)**

Lors de l'ajout d'une voiture, le SGBD doit **mettre à jour le nombre de voitures du parking auquel elle est affectée**

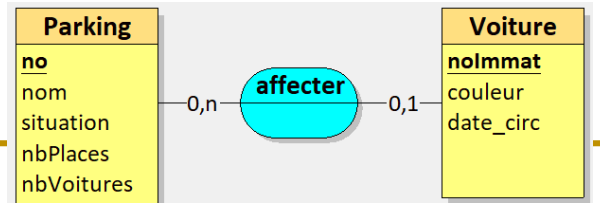
- Ex. de requête de mise à jour (« ajout d'une voiture ») ?

- Quel(s) trigger(s) faut-il créer ?

Rappel sur la structure d'un trigger :

```
{AFTER|BEFORE}
{DELETE|INSERT|UPDATE}
ON nom_relation

FOR EACH ROW
BEGIN
        Traitement à réaliser ;
END
```

Exemple Cas 1 (suite)



Parkings (no, nom, situation, nbPlaces, nbVoitures**)**
Voitures (noImmat, couleur, date_circ, parking#**)**

Lors de l'ajout d'une voiture, il faut **mettre à jour le nombre de voitures du parking auquel elle est affectée**

- Ex. de requête de mise à jour (« ajout d'une voiture ») ?

```
INSERT INTO Voitures (noImmat, couleur, date_circ, parking)
    VALUES ("AZ-781-EV", "bleu", "2020-06-12", 2);
```

- Quel(s) trigger(s) faut-il créer ?

```
{AFTER|BEFORE}
INSERT
ON Voitures

FOR EACH ROW
BEGIN
        -- mettre à jour l'attribut nbVoitures du parking auquel la
        -- voiture est affectée
END
```

Exemple Cas 1 (suite)

```
{AFTER|BEFORE}
INSERT
ON Voitures

FOR EACH ROW
BEGIN
        -- mettre à jour l'attribut nbVoitures du parking auquel la
        -- voiture est affectée
END
```

Traitement à réaliser

- **Mettre à jour l'attribut *nbVoitures* du parking auquel la voiture est affectée**
  - Exemple :

```
INSERT INTO Voitures (noImmat, couleur, date_circ, parking)
    VALUES ("AZ-781-EV", "bleu", "2020-06-12", 2);
```

- Comment réaliser le traitement ?
  - ➔ En SQL, comment mettre à jour l'attribut nbVoitures du parking 2 ?

Table Parkings :

no	nom	situation	nbPlaces	nbVoitures
1	Gouges	Nord-Est	4	0
2	Les roches	Près du centre	3	0

Exemple Cas 1 (suite)

```
{AFTER|BEFORE}
INSERT
ON Voitures

FOR EACH ROW
BEGIN
        -- mettre à jour l'attribut nbVoitures du parking auquel la
        -- voiture est affectée
END
```

Traitement à réaliser

- **Mettre à jour l'attribut *nbVoitures* du parking auquel la voiture est affectée**
  - Exemple de requête au cours de laquelle le traitement doit être effectué :

```
INSERT INTO Voitures (noImmat, couleur, date_circ, parking)
    VALUES ("AZ-781-EV", "bleu", "2020-06-12", 2);
```

- Comment réaliser le traitement ? ➔ **Avec une requête UPDATE** :
  - Cette requête doit être réalisée **dans le corps du trigger** …

Table Parkings :

no	nom	situation	nbPlaces	nbVoitures
1	Gouges	Nord-Est	4	0
2	Les roches	Près du centre	3	0

AFTER ou BEFORE ?

```
{AFTER|BEFORE}
{DELETE|INSERT|UPDATE}
ON nom_relation
FOR EACH ROW
BEGIN
        Instructions à réaliser ;
END
```

- **AFTER** : Le traitement du trigger est réalisé **après** la requête de mise à jour
- **BEFORE** : Le traitement du trigger est réalisé **avant** la requête de mise à jour

Le traitement peut être différent selon qu'il est réalisé AVANT ou APRES la requête de mise à jour

Exemple Cas 1 (suite)

```
{AFTER|BEFORE}
INSERT
ON Voitures

FOR EACH ROW
BEGIN
        -- mettre à jour l'attribut nbVoitures du parking auquel la
        -- voiture est affectée
END
```

```
INSERT INTO Voitures (noImmat, couleur, date_circ, parking)
    VALUES ("AZ-781-EV", "bleu", "2020-06-12", 2);
```

Traitement à réaliser par le trigger : un UPDATE sur la table Parkings

- 2 possibilités :
  - **On augmente de 1 l'attribut nbVoitures du parking 2** (même requête que l'ajout de la voiture ait lieu AVANT ou APRÈS)

AVANT le INSERT

```
BEFORE INSERT ON Voitures
UPDATE Parkings
    SET nbVoitures = nbVoitures + 1
    WHERE no = 2;
```

APRES le INSERT

```
AFTER INSERT ON Voitures
UPDATE Parkings
    SET nbVoitures = nbVoitures + 1
    WHERE no = 2;
```

  - **On compte le nombre voitures affectées au parking 2 dans la table Voitures** (il faut distinguer AVANT ou APRÈS : le nombre de voitures est différent !!!)

```
BEFORE INSERT ON Voitures
SELECT count(*) INTO @nb
    FROM Voitures WHERE parking = 2;
UPDATE Parkings
    SET nbVoitures = @nb + 1
    WHERE no = 2;
```

```
AFTER INSERT ON Voitures
SELECT count(*) INTO @nb
    FROM Voitures WHERE parking = 2;
UPDATE Parkings
    SET nbVoitures = @nb
    WHERE no = 2;
```

Exemple Cas 1 (suite)

```
{AFTER|BEFORE}
INSERT
ON Voitures

FOR EACH ROW
BEGIN
        -- mettre à jour l'attribut nbVoitures du parking auquel la
        -- voiture est affectée
END
```

```
INSERT INTO Voitures (noImmat, couleur, date_circ, parking)
    VALUES ("AZ-781-EV", "bleu", "2020-06-12", 2);
```

Le traitement retenu pour le trigger doit être applicable quelque soit le parking auquel la nouvelle voiture est affectée … (pas uniquement le parking 2)

Exemple :

```
UPDATE Parkings
    SET NbVoitures = NbVoitures + 1 WHERE no = 2;
```

Il FAUT utiliser les variables NEW et OLD dans le corps du trigger.
Cf. diapositives suivantes.

Accès aux attributs du tuple de la relation affectée par une mise à jour

- Dans le corps d'un trigger il est possible de faire référence aux **attributs du tuple ajouté, supprimé ou mis à jour** en utilisant les **variables NEW** et/ou **OLD**.
  - **OLD** est une variable qui contient les valeurs du tuple à supprimer (DELETE) ou à modifier (UPDATE) tel qu'il est dans la base avant sa suppression ou modification. La valeur d'un attribut du tuple (**OLD**.Attribut) peut être accédée/lue, mais **ne peut pas être modifiée**.
    - On **peut** écrire : IF OLD.Attribut = … ; -- test de la valeur dans la base
    - On **ne peut pas** écrire : ~~SET OLD.Attribut = … ;~~ -- affectation d'une valeur
  - **NEW** est une variable qui contient les valeurs du tuple à ajouter (INSERT) ou à modifier (UPDATE). La valeur d'un attribut du tuple (**NEW**.Attribut) peut toujours être accédée/lue**, et être modifiée mais uniquement dans un TRIGGER BEFORE** :
    - On **peut** écrire : IF NEW.Attribut = … ; -- test de la nouvelle valeur
    - On **peut** écrire (trigger BEFORE **uniquement**) :

      SET NEW.Attribut = … ; -- affectation d'une nouvelle valeur

Accès aux attributs du tuple de la relation affectée par une mise à jour

NB : **Au moment où un trigger est exécuté** :

- La **structure** des variables OLD et NEW est la même que celle de la table impactée par la mise à jour (mêmes attributs)
- Le **contenu** des variables OLD et NEW est le suivant :
  - Cas d'un trigger **INSERT** (OLD n'existe pas) :
    - **NEW** contient les valeurs du tuple à insérer dans la relation
  - Cas d'un trigger **DELETE** (NEW n'existe pas)
    - **OLD** contient les valeurs de tuple à supprimer de la relation (valeurs mémorisées dans la base)
  - Cas d'un trigger **UPDATE**
    - **OLD** contient les valeurs de tuple à modifier dans la relation (valeurs mémorisées dans la base)
    - **NEW** a les mêmes valeurs que OLD sauf pour les attributs modifiés par la requête (attributs sur lesquels porte le SET de la requête UPDATE) ou par le trigger lui-même
    - Ce sont les valeurs de NEW qui constitueront le contenu du tuple modifié dans la base.

Exemple Cas 1 (suite)

Parkings (no, nom, situation, nbPlaces, nbVoitures)
Voitures (noImmat, couleur, date_circ, parking#)

```
INSERT INTO Voitures (noImmat, couleur, date_circ, parking)
    VALUES ("AZ-781-EV", "bleu", "2020-06-12", 2);
```

Variable NEW :

noImmat	couleur	date_circ	parking
AZ-781-EV	bleu	2020-06-12	2

D'où le trigger :

```
AFTER
INSERT
ON Voitures

FOR EACH ROW
BEGIN
        -- mettre à jour l'attribut nbVoitures du parking auquel la
        -- voiture est affectée
        UPDATE Parkings SET nbVoitures = nbVoitures + 1
        WHERE no = NEW.parking;
END
```

Exercice

Parkings (no, nom, situation, nbPlaces, nbVoitures)
Voitures (noImmat, couleur, date_circ, parking#)

- Quel trigger mettre en place pour traiter
  1. le changement de parking d'une voiture ?
  2. la suppression d'une voiture ?

- Exemple de contenu de la base:

Parkings

no	nom	Situation	nbPlaces	nbVoitures
1	Gouges	Nord-Est	4	0
2	Les roches	Près du centre	3	1

Voitures

noImmat	couleur	date_circ	parking
AZ-781-EV	bleu	2020-06-12	2

1. La voiture AZ-781-EV est affectée au parking 1

Parkings

no	nom	Situation	nbPlaces	nbVoitures
1	Gouges	Nord-Est	4	1
2	Les roches	Près du centre	3	0

Voitures

noImmat	couleur	date_circ	parking
AZ-781-EV	bleu	2020-06-12	1

2. La voiture AZ-781-EV est supprimée

Parkings

no	nom	Situation	nbPlaces	nbVoitures
1	Gouges	Nord-Est	4	0
2	Les roches	Près du centre	3	0

Voitures

noImmat	couleur	date_circ	parking

Exercice

Parkings (no, nom, situation, nbPlaces, nbVoitures)
Voitures (noImmat, couleur, date_circ, parking#)

Voitures

noImmat	couleur	date_circ	parking
AZ-781-EV	bleu	2020-06-12	2

- Quel trigger mettre en place pour traiter
  1. le **changement de parking d'une voiture** ?

```
UPDATE Voitures SET parking = 1 WHERE noImmat = "AZ-781-EV";
```

Variable OLD :

noImmat	couleur	date_circ	parking

Variable NEW :

noImmat	couleur	date_circ	parking

Rappel sur la structure d'un trigger :

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

FOR EACH ROW
BEGIN
    Traitement à réaliser ;
END

Exercice

Parkings (no, nom, situation, nbPlaces, nbVoitures)
Voitures (noImmat, couleur, date_circ, parking#)

Voitures

noImmat	couleur	date_circ	parking
AZ-781-EV	bleu	2020-06-12	1

- Quel trigger mettre en place pour traiter
  2. la **suppression d'une voiture** ?

```
DELETE FROM Voitures
WHERE noImmat = "AZ-781-EV";
```

Variable OLD :

noImmat	couleur	date_circ	parking

Variable NEW :

noImmat	couleur	date_circ	parking

Rappel sur la structure d'un trigger :

```
{AFTER|BEFORE}
{DELETE|INSERT|UPDATE}
ON nom_relation

FOR EACH ROW
BEGIN
        Traitement à réaliser ;
END
```

FOR EACH ROW ?

```
{AFTER|BEFORE}
{DELETE|INSERT|UPDATE}
ON nom_relation
FOR EACH ROW
BEGIN
        Instructions à réaliser ;
END
```

- Comme vous le savez, il est possible d'ajouter | modifier | supprimer plusieurs tuples d'une table à l'aide d'une seule requête. Exemple :

```
INSERT INTO Voitures
        VALUES  ("AZ-781-EV", "bleu", "2020-06-12", 2),
                ("ER-975-TY", "rouge", NULL,1);
```

- Le corps du trigger (Cf. ci-dessous) sera exécuté **pour chaque tuple** (« FOR EACH ROW ») concerné par l'instruction.
- Exemple :

```
AFTER
INSERT
ON Voitures

FOR EACH ROW
BEGIN
        -- Mettre à jour l'attribut nbVoitures du parking auquel la
        -- voiture était affectée
        UPDATE Parkings SET nbVoitures = nbVoitures + 1
        WHERE no = NEW.parking;
END
```

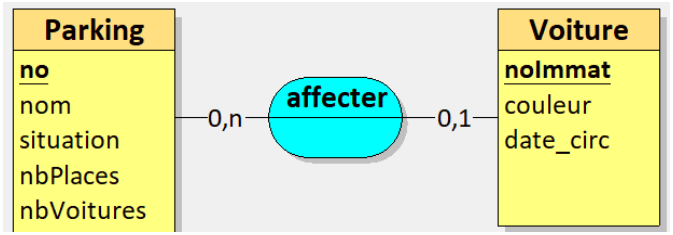
2 tuples sont insérés dans la table Voitures ; le trigger est déclenché pour chacun d'eux (soit 2 fois).

no	nom	situation	nbPlaces	nbVoitures
1	Gouges	Nord-Est	4	1
2	Les roches	Près du centre	3	1

Les triggers pour : Cas 2

- **Empêcher que certaines opérations entrainent un non-respect de contraintes**

- Exemple :



Règle 2
Pas plus de 3 parkings

  - La résidence n'a que 3 parkings …

Exemple Cas 2

Règle 1
Pas plus de 3 parkings

Parkings (no, nom, situation, nbPlaces, nbVoitures**)**
Voitures (nolmmat, couleur, date_circ, parking#**)**

Lors de l'ajout d'une parking, il faut **vérifier qu'il n'y aura pas plus de 3 parkings dans la relation Parkings**

- Ex. de requête de mise à jour (« ajout d'un parking ») ?

- Quel(s) trigger(s) faut-il créer ?

Rappel sur la structure d'un trigger :

```
{AFTER|BEFORE}
{DELETE|INSERT|UPDATE}
ON nom_relation

FOR EACH ROW
BEGIN
        Traitement à réaliser ;
END
```

Exemple Cas 2 (suite)

Règle 1
Pas plus de 3 parkings

Parkings (no, nom, situation, nbPlaces, nbVoitures**)**
Voitures (nolmmat, couleur, date_circ, parking#**)**

Lors de l'ajout d'une parking, il faut **vérifier qu'il n'y aura pas plus de 3 parkings dans la relation Parkings**

- Ex. de requête de mise à jour (« ajout d'un parking ») ?

```
INSERT INTO Parkings (nom, situation, nbPlaces)
VALUES   ("Gouges", "Nord-Est", 4), ("Les roches", "Près du centre", 3);
```

- Quel(s) trigger(s) faut-il créer ?

Rappel sur la structure d'un trigger :

BEFORE
INSERT
ON *Parkings*

FOR EACH ROW
BEGIN
Traitement à réaliser ;
END

Exemple Cas 2 (suite)

Règle 1
Pas plus de 3 parkings

- Comment vérifier la contrainte ?

  - Il faut connaître le nombre de parkings déjà présents dans la relation Parkings, d'où la requête SQL :

```
SELECT count(*) INTO @nbParkings FROM Parkings;
```

Ou bien :

```
SET @nbParkings = (SELECT count(*) FROM Parkings);
```

  - Si le nombre de parkings dans la table Parkings (variable @nbParkings) avant **l'insertion** du tuple, est de 3 alors STOP :
    - Il faut annuler l'ajout du nouveau tuple ! (sinon il y aurait 4 parkings dans la relation)
    - **Ce test doit être réalisé dans le corps du trigger**

Comment annuler une instruction du LMD ?

- Un trigger peut déclencher une erreur en fonction d'une condition de telle sorte à annuler l'instruction qui l'a déclenché.
- Nous savons tester une condition et exécuter des instructions en fonction de cette condition, grâce à l'instruction PL-SQL :

```
IF condition THEN
    instructions
 [ ELSE
    instructions ]
END IF;
```

- Pour déclencher une erreur, nous utiliserons l'instruction **SIGNAL** qui permet de renvoyer :
  - Un code d'erreur : nous utiliserons toujours le même : "45000"
  - Un message personnalisé (soyez explicite !)

"45000" est une chaine de caractères !

```
SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT = 'Message d'erreur !';
```

Exemple Cas 2 (suite)

- D'où le corps de trigger :

```
BEFORE
INSERT
ON Parkings
FOR EACH ROW
BEGIN
    SET @nbParkings = NULL;
    SELECT count(*) INTO @nbParkings FROM Parkings;

    IF (@nbParkings = 3) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Insertion refusée – Maximum 3 parkings !';
    END IF;

END
```

Avant que la requête INSERT sur la relation Parkings soit effectivement réalisée par le SGBD

Test de la contrainte

Stop ! Le traitement s'arrête. **Le INSERT est annulé**.

Pas de ELSE !

(ELSE …) Le traitement continue … et le INSERT est réalisé !
Ainsi, si la condition (@NbParkings = 3) **n'**est **pas** vérifiée, le traitement continue et **le INSERT sur la relation Parkings s'effectue normalement** (donc après le traitement du trigger)

Annulation d'une instruction du LMD : **For each row**

- Lorsqu'une mise à jour est annulée, toutes les mises à jour déjà effectuées pour la même requête (du fait de FOR EACH ROW) sont elles aussi annulées !
- Exemple : on suppose la table Parkings vide ; on exécute les deux requêtes suivantes :

```
INSERT INTO Parkings (nom, situation, nbPlaces)
VALUES   ("Gouges", "Nord-Est", 4);
INSERT INTO Parkings (nom, situation, nbPlaces)
VALUES   ("Les roches", "Près du centre", 3),
         ("Saint Benoit", NULL, 2),
         ("Leduc", "Près de l'entrée", 3);
```

➔ 1 tuple est ajouté à la table Parkings

Insertion refusée – Maximum 3 parkings !

C'est le 3ème tuple de la 2ème requête INSERT qui pose problème (car on a déjà 3 parkings) ; mais c'est la **requête INSERT entière qui est annulée ;**
AUCUN des trois tuples n'est ajouté dans la base !

➔ La table Parkings contient toujours 1 tuple après l'exécution de la requête INSERT

Syntaxe SQL pour créer/modifier un trigger

Comme pour les procédure et fonctions stockées : il faut changer le délimiteur (le **;** est utilisé dans les *instructions à réaliser* du trigger)

```
DELIMITER #
CREATE [OR REPLACE ] TRIGGER nom_trigger
{AFTER | BEFORE}
{DELETE | INSERT | UPDATE}
ON nom_relation
FOR EACH ROW
BEGIN
      Instructions à réaliser ;
END#
DELIMITER ;
```

Il y a un « espace » !

Convention de nommage des triggers (pour ce cours)

- Il ne peut exister qu'un seul trigger par combinaison moment/événement par relation.
  Donc pour une relation :
  - un seul trigger BEFORE INSERT,
  - un seul trigger AFTER INSERT,
  - un seul trigger BEFORE UPDATE,
  - etc.

info

Il existe :
- deux possibilités pour le moment d'exécution
- trois pour l'événement déclencheur
=> on peut définir au maximum six triggers par relation.

- Dans ce cours nous utiliserons la convention de **nommage des triggers** suivante :

  moment_evenement_relation.

- Exemple : Le trigger qui déclenchera un traitement avant l'insertion d'un tuple dans la relation *Parkings* aura donc pour nom :

  before_insert_Parkings

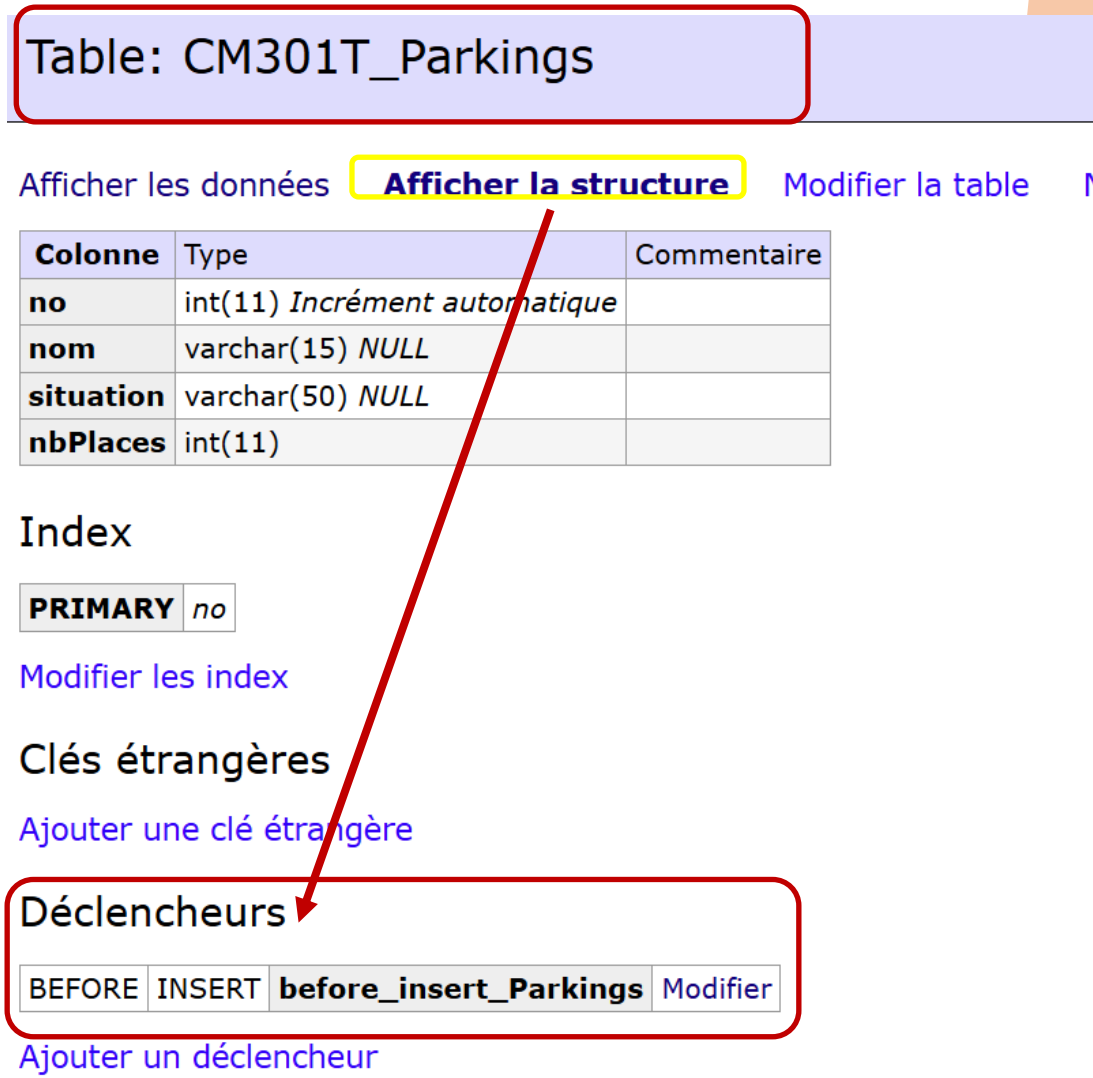
  ou un dérivé : ex. bef_ins_Parkings

Exemple SQL complet de trigger

```
DELIMITER //
-- les requêtes qui suivent doivent se terminer par //
CREATE OR REPLACE TRIGGER before_insert_Parkings
BEFORE INSERT
ON Parkings
FOR EACH ROW
BEGIN
    -- vérifier que la limite de 3 parkings n'a pas encore été atteinte
    SET @nbParkings = NULL;
    SELECT count(*) INTO @nbParkings FROM Parkings ;

    IF (@nbParkings = 3) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT =
                'Insertion refusée – Maximum 3 parkings ! ';
    END IF;

END//

DELIMITER ;
-- les requêtes qui suivent doivent se terminer par ;
```

Visualiser un trigger dans Adminer

- Dans Adminer, les triggers sont visibles et modifiables depuis la page **affichant la structure de la relation** :



Exporter les triggers avec Adminer

- Pour exporter le code SQL de création des triggers associés aux relations de votre base de données, pensez à cocher l'option « Déclencheurs »

Mise en œuvre

```
-- Création de la table Parkings
CREATE TABLE CM301T_Parkings (
        no INT AUTO_INCREMENT PRIMARY KEY,
        nom VARCHAR(15),
        situation VARCHAR(20),
        nbPlaces INT NOT NULL,
        nbVoitures INT DEFAULT 0) ;

-- Création du trigger before_insert_Parkings
-- Cf. Diapo précédente

-- Ajouts de tuples (des parkings) dans la relation CM301T_Parkings

INSERT INTO CM301T_Parkings (nom, situation, nbPlaces)
    VALUES ("Gouges", "Nord-Est", 4);
INSERT INTO CM301T_Parkings (nom, situation, nbPlaces)
    VALUES ("Les roches", "Près du centre", 3);
INSERT INTO CM301T_Parkings (nom, situation, nbPlaces)
    VALUES ("Saint Benoit", NULL, 2),
           ("Leduc", "Près de l'entrée", 3);
```

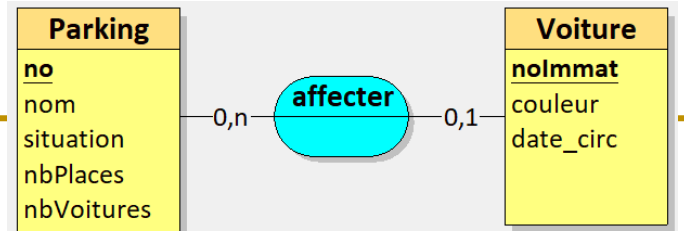
Démo
en direct

Combien y a-t-il de tuples dans CM301T_Parkings ?

Combien y a-t-il de tuples dans CM301T_Parkings ?

Combien y a-t-il de tuples dans CM301T_Parkings ?

Exercice



Parkings (no, nom, situation, nbPlaces, nbVoitures)
Voitures (noImmat, couleur, date_circ, parking#)

- Le nombre de voitures par parking est limité.

- Comment assurer cette contrainte avec des triggers ?

1. Affectation d'un parking à une nouvelle voiture
2. Changement de parking d'une voiture
3. Diminution du nombre de places d'un parking auquel des voitures sont déjà affectées

Synthèse sur les triggers

- Les triggers servent donc à maintenir la cohérence pour des données calculées, ou dupliquées, mais ils peuvent aussi servir à d'autre choses comme nous l'avons vu :
  - Effectuer des contrôles et refuser la mise à jour de la base en cas d'anomalie
  - Déclencher des traitements complémentaires sur d'autres relations
  - Accéder à des fonctions système (par ex. imprimer un message quand une contrainte d'intégrité est violée)
  - Mémoriser un historique des modifications
  - etc.

Quelques commandes utiles

- Afficher les contraintes définies pour une relation (CHECK) :
  SHOW CREATE TABLE <nom de la relation>;
- Affichage de tous les triggers :
  SHOW TRIGGERS ;
- Suppression d'un trigger :
  DROP TRIGGER before_insert_Parkings **;**

Sources

- https://mariadb.com/kb/en/
- https://dev.mysql.com/doc/refman/8.0/en/
- https://sql.sh
- https://grafikart.fr/tutoriels/procedures-triggers-fonctions-593