ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR

Membre de HONORIS UNITED UNIVERSITIES

École Marocaine des Sciences de l'Ingénieur

Filière: IIR

Les Bases de Données

Prof. Zakaria KHATAR

Triggers en PL/SQL
(Déclencheurs)

1) Les Triggers :

Définition :

Un trigger en PL/SQL (ou Déclencheur en français) est un programme qui se déclenche automatiquement en réponse à certaines opérations dans une base de données. Ces opérations incluent **les insertions (INSERT), les mises à jour (UPDATE)** et **les suppressions de données (DELETE)** dans une table. Le trigger est activé par ces actions spécifiques et exécute le code qu'il contient. Ce code est défini lors de la création du trigger, et il peut effectuer diverses tâches, comme valider des données ou maintenir l'intégrité de la base de données.

1) Les Triggers :

Types de Triggers :

- **Triggers BEFORE :** Exécutés avant des actions comme l'insertion, la mise à jour ou la suppression dans une table, ils sont utilisés pour vérifier ou modifier les données en amont.
- **Triggers AFTER :** Activés après les opérations d'insertion, de mise à jour ou de suppression, ils servent à effectuer des tâches en réaction aux changements apportés.
- **Triggers INSTEAD OF :** Employés spécifiquement avec les vues pour gérer les opérations que les vues ne peuvent pas exécuter directement, comme l'insertion ou la mise à jour.

1) Les Triggers :

La syntaxe :

```
CREATE [OR REPLACE] TRIGGER nom_trigger
    BEFORE | AFTER | INSTEAD OF événement ON nom_table_ou_vue
    FOR EACH ROW
    WHEN (condition)
Declare
        -- Déclaration de Variables
BEGIN
        -- instructions PL/SQL
END;
```

1) Les Triggers :

Composants de la Syntaxe :

- **CREATE [OR REPLACE]** : Commence la définition d'un nouveau trigger. "OR REPLACE" est optionnel et permet de remplacer un trigger existant du même nom.
- **nom_du_trigger** : Nom du trigger que vous créez.
- **BEFORE|AFTER|INSTEAD OF** : Type de trigger. Utilisez BEFORE ou AFTER pour des opérations sur les tables, et INSTEAD OF pour des opérations sur les vues.
- **événement** : L'événement qui déclenche le trigger, comme INSERT, UPDATE, ou DELETE.
- **nom_table_ou_vue** : La table ou la vue sur laquelle le trigger est appliqué.
- **FOR EACH ROW** : Spécifie que le trigger doit s'exécuter pour chaque ligne affectée par l'événement. Omettez cette clause pour un trigger au niveau de la déclaration (statement-level trigger).
- **WHEN (condition)** : Une condition optionnelle pour limiter quand le trigger s'exécute. Disponible uniquement pour les triggers de type ROW.
- **DECLARE** : Section pour déclarer des variables. Cette partie est facultative et utilisée seulement si vous avez besoin de déclarer des variables dans votre trigger.
- **BEGIN ... END;** : Le bloc PL/SQL qui contient le code à exécuter lorsque le trigger est activé.

1) Les Triggers :

Notez que la clause **OR REPLACE,** la clause **WHEN** et la clause **DECLARE** sont optionnelles et peuvent être supprimées si elles ne sont pas nécessaires.

1) Les Triggers :

Exemple 1 :

Le déclencheur suivant affiche le message 'Modification Effectuée, Merci' après chaque modification de la table des `employees`.

```
CREATE OR REPLACE TRIGGER msg_update
        AFTER UPDATE ON employees
        FOR EACH ROW
BEGIN
        DBMS_OUTPUT.PUT_LINE('Modification Effectuée, Merci');
END;
```

1) Les Triggers :

Exemple 2 :

Le déclencheur suivant affiche le message 'Supression Effectuée, Merci' après chaque modification de la table des `employees`.

```
CREATE OR REPLACE TRIGGER msg_suppression
        AFTER DELETE ON employees
        FOR EACH ROW
BEGIN
        DBMS_OUTPUT.PUT_LINE('Supression Effectuée, Merci');
END;
```

1) Les Triggers :

Les clauses **NEW** et **OLD** sont utilisées dans les triggers de PL/SQL pour se référer aux **nouvelles valeurs et aux anciennes valeurs** d'une ligne qui est modifiée ou insérée dans une table.

La clause **NEW** se réfère aux nouvelles valeurs qui ont été insérées ou qui ont été modifiées dans la table, alors que la clause **OLD** se réfère aux anciennes valeurs de la ligne avant qu'elle soit mise à jour.

Syntaxe :

Syntaxe de la clause NEW :

:NEW.nom_colonne

Syntaxe de la clause OLD :

:OLD.nom_colonne

1) Les Triggers :

Exemple 1 : Cet exemple de trigger vérifie si le salaire d'un employé a été augmenté, diminué ou inchangé lorsqu'une modification est effectuée sur la table "employees", et affiche le message correspondant.

```
CREATE OR REPLACE TRIGGER update_salary
    BEFORE UPDATE ON employees
    FOR EACH ROW
BEGIN
      IF :NEW.salary > :OLD.salary THEN
            DBMS_OUTPUT.PUT_LINE('le salaire a été augmenté');
      ELSIF :NEW.salary < :OLD.salary THEN
            DBMS_OUTPUT.PUT_LINE('le salaire a été diminué');
      ELSE
            DBMS_OUTPUT.PUT_LINE('Pas d''opération sur le salaire');
      END IF;
END;
```

1) Les Triggers :

Exemple 2 : Cet exemple de trigger s'active avant l'insertion d'une nouvelle ligne dans une table (employees) pour attribuer une valeur par défaut à une colonne si elle est laissée vide.

```
CREATE OR REPLACE TRIGGER before_insert_employee
    BEFORE INSERT ON employees
    FOR EACH ROW
BEGIN
      IF :NEW.job_id IS NULL THEN
            :NEW.job_id := 'INCONNU';
      END IF;
END;
```

2) Les Vues :

Définition : Une vue est une **table virtuelle** dans une base de données, créée à partir d'une requête SQL. Elle permet de voir et d'interagir avec les données d'une manière personnalisée sans les modifier directement.

Syntaxes :

```
CREATE OR REPLACE VIEW nom_de_la_vue AS
    SELECT colonne1, colonne2, ...
        FROM table_source
            WHERE nouvelle_condition;
```

Une vue en SQL agit comme une table standard pour les requêtes de lecture, mais contrairement à une table réelle, elle ne permet généralement pas la modification directe de ses données, car elle représente simplement le résultat d'une requête d'interrogation.

2) Les Vues :

Exemple : La création d'une vue nommée **employee_details** qui combine les informations des deux tables **employees** et **departments** :

```
CREATE OR REPLACE VIEW employee_details AS
    SELECT first_name, last_name, department_name
        FROM employees
            INNER JOIN departments USING(department_id);
```

Cette vue employee_details affiche le prénom de l'employé, son nom et le nom de son département.

3) Les Triggers Instead Of :

Exemple 3 : Cet exemple de trigger **INSTEAD OF sur une Vue** qui permet d'effectuer une insertion sur une vue d'une seule table.
Supposons que vous ayez une vue employee_details qui est une représentation personnalisée de la table employees. Vous voulez que les insertions sur cette vue soient reflétées dans la table employees.

```
CREATE OR REPLACE TRIGGER insert_view_employee
    INSTEAD OF INSERT ON employee_details
    FOR EACH ROW
BEGIN
        INSERT INTO employees (last_name, first_name)
            VALUES (:NEW.last_name, :NEW.first_name);
        INSERT INTO depatments (department_name)
            VALUES (:NEW.department_name);
END;
```

3) Les Triggers Instead Of :

Exemple : Trigger INSTEAD OF pour la Mise à Jour

Supposons maintenant que vous voulez permettre la mise à jour du nom d'un employé directement à travers la vue. Vous pouvez créer un trigger **INSTEAD OF UPDATE** pour cela :

```
CREATE OR REPLACE TRIGGER update_employee_email
INSTEAD OF UPDATE ON employee_details      < - - - - - la vue crée
FOR EACH ROW
BEGIN
   UPDATE employees
        SET last_name = :NEW.last_name
            WHERE employee_id = :NEW.employee_id;
END;
```

4) Désactivation et réactivation des Triggers :

En PL/SQL, la gestion des triggers, notamment leur désactivation et réactivation, ainsi que l'affichage se font à l'aide des commandes suivantes:

Action	Commande
Désactivation de Trigger	**ALTER TRIGGER** nom_du_trigger **DISABLE;**
Réactivation de Trigger	**ALTER TRIGGER** nom_du_trigger **ENABLE;**
Affichage des Triggers	**SELECT** trigger_name **FROM** user_triggers**;**