PL/SQL : Les Triggers

Les Triggers

➢ Qu'est-ce qu'un trigger ?

Un **trigger** est un bloc de code PL/SQL qui s'exécute automatiquement en réponse à un événement spécifique sur une table, une vue ou la base de données. Les triggers sont déclenchés par des événements comme des opérations DML (INSERT, UPDATE, DELETE), des instructions DDL (CREATE, ALTER, DROP) ou des événements système (LOGON, LOGOFF, etc.).

Caractéristique clé : Contrairement aux procédures ou fonctions, un trigger est **automatique** et ne nécessite pas d'appel explicite.

Les Triggers: Utilités

1. **Automatisation des tâches :** Les triggers permettent d'automatiser des actions spécifiques, comme
    1. Maintenir un journal des modifications sur une table.
    2. Mettre à jour des valeurs dérivées (exemple : recalculer un total après une insertion).
2. **Maintien de l'intégrité des données :** Les triggers peuvent appliquer des règles supplémentaires pour garantir la cohérence des données, par exemple :
    1. Empêcher la suppression d'une ligne si elle est référencée ailleurs.
    2. Vérifier qu'une valeur respecte des contraintes personnalisées.
3. **Mise en place de règles métier :** Ils permettent d'implémenter des règles spécifiques au contexte métier directement au niveau de la base de données, par exemple :
    1. Calculer automatiquement une prime lors de l'ajout d'un employé si son salaire dépasse un certain seuil.
    2. Bloquer les modifications sur des colonnes sensibles en fonction de l'utilisateur connecté.

Les Triggers

➢ Comparaison avec les procédures et fonctions stockées

Aspect	Trigger	Procédure/Fonction
Appel	S'exécute automatiquement lorsqu'un événement survient.	Doit être appelée explicitement dans le code PL/SQL.
Utilisation	Réagit aux changements dans la base de données ou aux événements.	Utilisé pour encapsuler et exécuter une logique spécifique.
Portée principale	Lié à des événements spécifiques sur des tables ou la base.	Peut être utilisé indépendamment dans différents contextes.
Impact sur les données	Peut lire ou modifier les données affectées par l'événement.	Peut manipuler des données mais nécessite un appel direct.

Les Triggers

➢ **Les types de triggers selon le type d'événement déclencheur**

- **DML Triggers** (Data Manipulation Language) : Déclenchés par INSERT, UPDATE, DELETE.
- **DDL Triggers** (Data Definition Language) : Déclenchés par CREATE, ALTER, DROP.
- **Triggers sur les événements de base de données** : Déclenchés par des événements tels que LOGON, LOGOFF, STARTUP, SHUTDOWN.

Les Triggers

➢ Les types de triggers selon le moment où ils s'exécutent

- **Avant (BEFORE) :**
  1. S'exécute **avant** qu'un événement ne soit effectué sur une table ou une vue.
  2. Utilisé pour valider ou modifier des données avant qu'elles ne soient insérées, mises à jour ou supprimées.

  Exemple : Vérifier qu'un champ obligatoire n'est pas vide avant un INSERT.
- **Après (AFTER) :**
  1. S'exécute **après** qu'un événement a eu lieu.
  2. Utilisé pour effectuer des actions qui nécessitent que les données soient déjà modifiées.

  Exemple : Ajouter une ligne dans une table de journalisation après une modification.
- **Au lieu de (INSTEAD OF) :**
  1. S'exécute **à la place** d'une opération DML sur une vue (principalement utilisé pour les vues complexes non directement modifiables).
  2. Permet de définir comment une opération sur une vue doit être traduite en opérations sur les tables sous-jacentes.

  Exemple : Insérer une donnée dans plusieurs tables sous-jacentes lorsqu'une opération INSERT est effectuée sur une vue.

Les Triggers

➢ **Syntaxe**

```
CREATE [OR REPLACE] TRIGGER nom_du_trigger
{BEFORE | AFTER | INSTEAD OF} événement
ON table
[FOR EACH ROW] -- optionnel
[WHEN condition] -- optionnel
DECLARE
    -- Déclarations optionnelles
BEGIN
    -- Logique PL/SQL
END;
```

Les Triggers

➢ Exemple

```
CREATE OR REPLACE TRIGGER
ajoutClient
BEFORE INSERT ON clients
BEGIN
IF USER != ‘EMSI' THEN
RAISE_APPLICATION_ERROR (-20001, 'Utilisateur
interdit');
END IF;
END ajoutClient;
```

Les Triggers

Les **niveaux d'exécution d'un déclencheur** (trigger) en PL/SQL définissent la portée et la manière dont le trigger réagit à un événement. Il existe deux principaux niveaux d'exécution pour les triggers en PL/SQL.

1. Triggers Row-Level: Un **trigger Row-Level** s'exécute **une fois pour chaque ligne affectée** par l'opération qui déclenche le trigger.

Caractéristiques :

- Exécuté **pour chaque ligne** concernée par l'opération DML (INSERT, UPDATE, DELETE).
- Accès aux valeurs de la ligne en cours de traitement via les variables **:NEW** et **:OLD** :
  - :NEW : Représente les nouvelles valeurs de la ligne (par exemple, après un INSERT ou un UPDATE).
  - :OLD : Représente les anciennes valeurs de la ligne (par exemple, avant un DELETE ou un UPDATE).
- Nécessite l'utilisation de la clause FOR EACH ROW dans la déclaration du trigger.

Les Triggers

Exemple : Un trigger qui vérifie si le salaire d'un employé reste dans une plage autorisée avant une mise à jour :

```
CREATE OR REPLACE TRIGGER TRG_CHECK_SALARY
BEFORE UPDATE ON EMPLOYE
FOR EACH ROW
BEGIN
    IF :NEW.SALAIRE < 1000 OR :NEW.SALAIRE > 5000 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Salaire hors limites
autorisées');
    END IF;
END; /
```

2. Triggers Statement-Level

Un **trigger Statement-Level** s'exécute **une seule fois par opération DML**, quelle que soit la quantité de lignes affectées.

Caractéristiques :

- Exécuté **une fois** par événement (INSERT, UPDATE, DELETE), même si plusieurs lignes sont affectées.
- **Pas d'accès aux variables :NEW et :OLD**, car le trigger agit au niveau de l'opération et non au niveau des lignes individuelles.
- Par défaut, tous les triggers sont **Statement-Level** si la clause FOR EACH ROW n'est pas spécifiée.

Exemple : Un trigger qui enregistre le nombre de modifications effectuées sur une table :

```
CREATE OR REPLACE TRIGGER TRG_LOG_UPDATE
AFTER UPDATE ON EMPLOYE
BEGIN
    INSERT INTO AUDIT_LOG (TABLE_NAME, ACTION,
DATE_OPERATION)
    VALUES ('EMPLOYE', 'UPDATE', SYSDATE);
END;
/
```

Les Triggers

Valeurs des attributs :NEW et :OLD. Disponibilité selon le type d'opération :

- **INSERT :** L'attribut :OLD **n'est pas accessible**, car il n'existe pas d'ancienne valeur pour une nouvelle insertion.
- **UPDATE :** Les attributs :OLD **et** :NEW **sont accessibles** :
    - :OLD contient les valeurs **avant la mise à jour**.
    - :NEW contient les nouvelles valeurs **après la mise à jour**.
- **DELETE :** L'attribut :NEW **n'est pas accessible**, car il n'existe plus de nouvelle valeur après la suppression d'une ligne.

Important : Les attributs :NEW et :OLD sont **uniquement disponibles dans un déclencheur de niveau "enregistrement" (Row-Level Trigger)**, défini avec la clause FOR EACH ROW.

1. INSERT : Initialisation d'une colonne

Créer un déclencheur qui initialise automatiquement une colonne DATE_CREATION à la date et l'heure actuelles lorsqu'un nouvel enregistrement est inséré dans la table CLIENT.

```
CREATE OR REPLACE TRIGGER TRG_CLIENT_DATE_CREATION
BEFORE INSERT ON CLIENT
FOR EACH ROW
BEGIN
    :NEW.DATE_CREATION := SYSDATE;
END;/
```

2. UPDATE : Vérification et contrôle des données

Créer un déclencheur qui empêche de réduire le salaire d'un employé lors d'une mise à jour dans la table EMPLOYE.

```
CREATE OR REPLACE TRIGGER TRG_EMPLOYE_CHECK_SALARY
BEFORE UPDATE ON EMPLOYE
FOR EACH ROW
BEGIN
    IF :NEW.SALAIRE < :OLD.SALAIRE THEN
        RAISE_APPLICATION_ERROR(-20001, 'Le salaire ne peut pas être
réduit.');
    END IF;
END;/
```

3. DELETE : Journalisation des suppressions

Créer un déclencheur qui enregistre dans une table de log les informations sur les clients supprimés de la table CLIENT.

```
CREATE OR REPLACE TRIGGER TRG_CLIENT_DELETE_LOG
AFTER DELETE ON CLIENT
FOR EACH ROW
BEGIN
    INSERT INTO CLIENT_LOG (ID_CLIENT, NOM, ACTION, DATE_OPERATION)
    VALUES (:OLD.ID_CLIENT, :OLD.NOM, 'DELETE', SYSDATE);
END;
/
```

➢ VALEURS DES ATTRIBUTS: LA CLAUSE REFERENCING

La **clause REFERENCING** est utilisée dans un déclencheur pour donner des **alias** alternatifs aux mots-clés :NEW et :OLD. Cela permet d'améliorer la lisibilité ou de résoudre des conflits éventuels avec d'autres noms dans le déclencheur.

```
CREATE OR REPLACE TRIGGER nom_trigger
{BEFORE | AFTER | INSTEAD OF} événement
ON nom_table
[FOR EACH ROW]
REFERENCING { NEW AS alias_new | OLD AS alias_old }
BEGIN
    -- Logique PL/SQL
END;
```

Exemple 1 : Utilisation pour une meilleure lisibilité

Un déclencheur qui vérifie qu'un nouvel employé a un salaire supérieur au salaire minimum.

```
CREATE OR REPLACE TRIGGER TRG_SALARY_CHECK
BEFORE INSERT OR UPDATE ON EMPLOYE
FOR EACH ROW
REFERENCING NEW AS NOUVELLE OLD AS ANCIENNE
BEGIN
    IF :NOUVELLE.SALAIRE < 1500 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Le salaire doit être
supérieur à 1500.');
    END IF;
END;/
```

Les Triggers

Exemple 2 : Résolution d'un conflit de noms

Supposons qu'une table contient une colonne nommée NEW. Cela pourrait entrer en conflit avec le mot-clé :NEW. Utiliser la clause REFERENCING permet d'éviter ce problème.

```
CREATE TABLE PRODUIT( ID_PRODUIT NUMBER PRIMARY KEY,  NEW VARCHAR2(100),PRIX NUMBER);
```

```
CREATE OR REPLACE TRIGGER TRG_PRODUIT_AUDIT
BEFORE UPDATE ON PRODUIT
FOR EACH ROW
REFERENCING NEW AS NV OLD AS OV
BEGIN
    -- Utilisation des alias NV et OV pour éviter le conflit avec la colonne
'NEW'
    INSERT INTO PRODUIT_LOG (ID_PRODUIT, CHAMP, ANCIENNE_VALEUR,
NOUVELLE_VALEUR)
    VALUES (:OV.ID_PRODUIT, 'PRIX', :OV.PRIX, :NV.PRIX);
END;
/
```

Les Triggers

Le déclenchement conditionnel permet de définir une logique spécifique dans un **trigger** en fonction de certaines **conditions**. Cela permet de restreindre l'exécution du code à des cas particuliers au lieu d'exécuter le trigger pour toutes les lignes ou toutes les opérations.

En PL/SQL, le déclenchement conditionnel peut être mis en place de deux manières principales :

1. Avec la clause WHEN.

2. Avec des conditions dans le corps du trigger.

```
CREATE OR REPLACE TRIGGER nom_trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE} ON table
[FOR EACH ROW]
WHEN (condition)
BEGIN
    -- Logique du trigger
END;/
```

Les Triggers

Exemple : Un trigger qui ne s'exécute que si le **salaire d'un employé est supérieur à 5000** après une mise à jour.

```
CREATE OR REPLACE TRIGGER TRG_CHECK_HIGH_SALARY
AFTER UPDATE ON EMPLOYE
FOR EACH ROW
WHEN (:NEW.SALAIRE > 5000)
BEGIN
    INSERT INTO AUDIT_LOG (ID_EMPLOYE, ACTION, DATE_OPERATION)
    VALUES (:NEW.ID_EMPLOYE, 'SALARY > 5000', SYSDATE);
END;
/
```

2. Conditions dans le Corps du Trigger

Une autre méthode consiste à écrire des **conditions explicites** directement dans le corps du trigger. Cela permet une logique conditionnelle plus complexe.

Exemple : Un trigger qui bloque les suppressions d'un employé si son poste est "Manager".

```
CREATE OR REPLACE TRIGGER TRG_BLOCK_DELETE_MANAGER
BEFORE DELETE ON EMPLOYE
FOR EACH ROW
BEGIN
    IF :OLD.POSTE = 'Manager' THEN
        RAISE_APPLICATION_ERROR(-20002, 'Impossible de supprimer un Manager.');
    END IF;
END;
/
```

Les Triggers

Les **fonctions prédicats** en PL/SQL sont des fonctions système utilisées dans les déclencheurs (triggers) pour identifier **l'opération DML** (INSERT, UPDATE, DELETE) qui a déclenché l'exécution du trigger.

Liste des Fonctions Prédicats

1. **INSERTING** : Retourne TRUE si l'opération déclenchant le trigger est un INSERT.
2. **UPDATING** : Retourne TRUE si l'opération déclenchant le trigger est un UPDATE.
3. **DELETING** : Retourne TRUE si l'opération déclenchant le trigger est un DELETE.

Les Triggers

```
CREATE OR REPLACE TRIGGER TRG_AUDIT_EMPLOYE
AFTER INSERT OR UPDATE OR DELETE ON EMPLOYE
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO AUDIT_LOG (ID_LOG, ID_EMPLOYE, OPERATION, DATE_OPERATION)
        VALUES (SEQ_AUDIT_LOG.NEXTVAL, :NEW.ID_EMPLOYE, 'INSERT', SYSDATE);

    ELSIF UPDATING THEN
        INSERT INTO AUDIT_LOG (ID_LOG, ID_EMPLOYE, OPERATION, DATE_OPERATION)
        VALUES (SEQ_AUDIT_LOG.NEXTVAL, :NEW.ID_EMPLOYE, 'UPDATE', SYSDATE);

    ELSIF DELETING THEN
        INSERT INTO AUDIT_LOG (ID_LOG, ID_EMPLOYE, OPERATION, DATE_OPERATION)
        VALUES (SEQ_AUDIT_LOG.NEXTVAL, :OLD.ID_EMPLOYE, 'DELETE', SYSDATE);
    END IF;
END;
/
```

```
CREATE OR REPLACE TRIGGER TRG_VALIDATE_EMPLOYE
BEFORE INSERT OR UPDATE OR DELETE ON EMPLOYE
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        IF :NEW.SALAIRE < 1000 THEN
            RAISE_APPLICATION_ERROR(-20001, 'Le salaire doit être supérieur à
1000 pour un nouvel employé.’); END IF;

    ELSIF UPDATING THEN
        IF :NEW.SALAIRE < :OLD.SALAIRE THEN
            RAISE_APPLICATION_ERROR(-20002, 'Impossible de réduire le salaire
d’un employé.’);      END IF;

    ELSIF DELETING THEN
        IF :OLD.POSTE = 'Manager' THEN
            RAISE_APPLICATION_ERROR(-20003, 'Impossible de supprimer un
Manager.');
        END IF;
    END IF;
END;/
```

Les Triggers

- **ALTER TRIGGER nom-déclencheur DISABLE;**
- **ALTER TABLE nom-table DISABLE ALL TRIGGERS;**
- **ALTER TRIGGER nom-déclencheur ENABLE;**
- **ALTER TABLE nom-table ENABLE ALL TRIGGERS;**