Université Mohammed V - Rabat
Ecole Mohammadia d'Ingénieurs

Filière Réseaux et Télécom

Bases de Données

Programmation PL/SQL

Mr N.EL FADDOULI

2023-2024

Triggers (déclencheurs)

- Définition
- Utilité
- Structure d'un trigger
- Exemples

Définition

- Trigger / Déclencheur:
  - **Action** ou ensemble d'actions déclenchée(s) **automatiquement** lorsqu'une **condition** se trouve satisfaite après l'apparition d'un **événement**.
- Un déclencheur est une règle **ECA**
  - **Evénement** = **DML** (Update, Delete, Insert) , **DDL** (Create, Alter, …), **Système** (Startup, Shutdown), **Utilisateur** (Connexion, déconnexion)
  - **Condition** = expression logique vraie ou fausse, optionnelle
  - **Action** = bloc (procédure) exécuté lorsque la condition est vérifiée.
- Implémentation dans les SGBD:
  - Avec ORACLE, les événements sont prédéfinies et les conditions et les actions sont spécifiées par le langage procédural PL/SQL.
  - Avec DB2, le langage est le C

Utilité

- Maintien de contraintes d'intégrité.

  Exemple 1: Imposer que le salaire ∈[5000, 30000]

  Exemple 2: Etudiant (matricule, nom, …., # Codfiliere),

  Filière(Codfiliere,….)

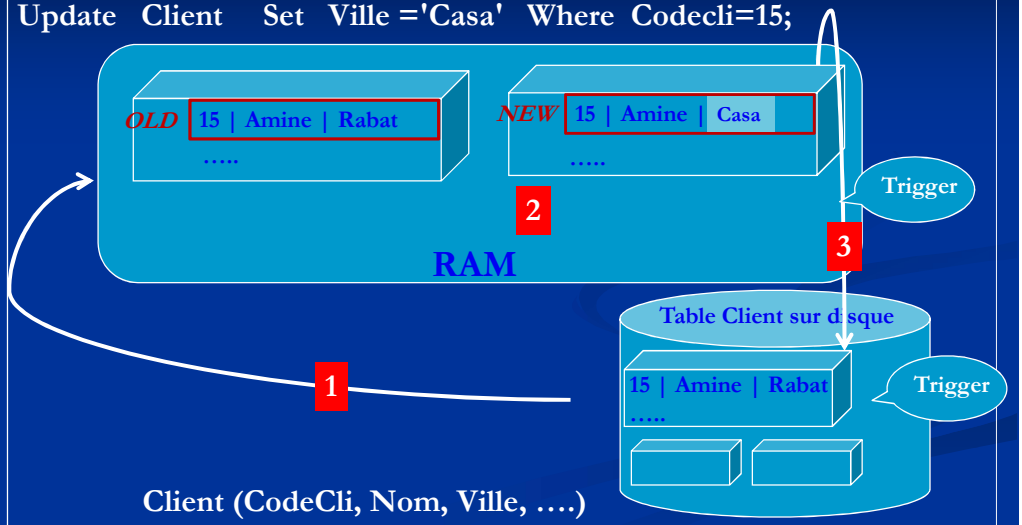
  Chambre(Codchambre, # matricule1, # matricule2, étage, …)

  Chaque fois qu'un étudiant est ajouté, on l'affectera à une chambre occupé par un étudiant d'une filière différente ou libre (matricule1 = maatricule2= NULL)
- Propagation des MAJ dans une base de données distribuée
- Sécurité
- .....

Structure d'un trigger DML (2)

Comment une requête de MAJ est exécutée?

Update Client Set Ville ='Casa' Where Codecli=15;



Client (CodeCli, Nom, Ville, ….)

Structure d'un trigger DML (1)

```
CREATE [ OR REPLACE ] TRIGGER nom_trigger
    {BEFORE | AFTER | INSTEAD OF}
    {INSERT | DELETE | UPDATE [OF liste de colonnes ] }   // événement
    ON table
    [ FOR EACH ROW ]    [ FOLLOWS un_autre_trigger ]  [ ENABLE | DISABLE]
    [ WHEN (condition_déclenchement) ]  // condition
    DECLARE
    ……
    BEGIN
    …….                  //Actions avec les données
    EXCEPTION
    ……..
    END ;
    /
```

Structure d'un trigger DML (3)

- Exemple: Suppression d'un client

  → On supprime toutes ses commandes

```
CREATE TRIGGER suppclient
BEFORE DELETE ON CLIENT
 FOR EACH ROW
BEGIN
DELETE FROM COMMANDE WHERE Codecli= :OLD.Codecli;
dbms_output.put_line ('Ligne supprimée');
END;
/
```

Une ligne de CLIENT avant sa suppression

- Ce trigger sera déclenché suite à une requête de suppression, par exemple:

  Delete From Client Where CodeCli Between 12 and 45

Structure d'un trigger DML (4)

- Exemple: Modification de la ville d'un client

  → Convertir la ville en majuscules

```
CREATE TRIGGER majclient
BEFORE Update (Ville) ON CLIENT
 FOR EACH ROW
BEGIN
    :NEW.Ville := Upper (:NEW.Ville ) ;
END;
/
```

Une ligne de CLIENT après sa modification dans la **RAM**

- Ce trigger sera déclenché suite à une requête de modification, par exemple:

  Update Client Set Ville='casa' **Where** CodeCli=15 ;

Structure d'un trigger DML (5)

Une structure de trigger est composée de trois parties:

- ✓ Un événement déclencheur **E**: action externe sur une table ou sur un tuple qui déclenche le trigger.
- ✓ Une condition de déclenchement **C**: C'est une expression booléenne
- ✓ Une action du trigger **A** : c'est un bloc PL/SQL

Remarques :

1. Lorsqu'un trigger est lancé sur le serveur et qui se termine **sans** traitement d'exception, l'événement qui l'a déclenché se poursuit correctement.
2. Deux types de triggers peuvent être définis:
   1. **Trigger d'énoncé** : C'est un trigger lancé une seule fois.
   2. **Tigger de tuple**: trigger exécuté autant de fois qu'il y a de tuples à insérer, à modifier ou à supprimer
3. Par défaut un trigger est actif, il peut cependant être désactivé:

ALTER TRIGGER nom_trigger **DISABLE** ;
ALTER TRIGGER nom_trigger **ENABLE** ;

Structure d'un trigger DML (6)

Remarques :

1. Dans les triggers de tuples (**FOR EACH ROW**), on peut manipuler les valeurs traitées, directement **en mémoire**:
   - ✓ **: old**.attribut : ancienne valeur (**DELETE** | **UPDATE**)
   - ✓ **: new**.attribut : nouvelle valeur (**INSERT** | **UPDATE**)

Exemple: Trigger pour suppression et sauvegarde dans une table de journal

```
CREATE OR REPLACE TRIGGER suppression_ligne
  BEFORE DELETE ON produit
  FOR EACH ROW
  BEGIN
INSERT INTO Journal VALUES (:old.codprod, :old.libelle, :old.prix,
                            :old.qte) ;
  END;
```

Delete From produit Where prix<10;

Exemple 1: Lorsqu'une augmentation du *prix Unitaire* (PU)d'un *Produit* est tentée, il faut limiter l'augmentation à 10% du prix en cours

```
CREATE OR REPLACE TRIGGER BORNER_AUGMENTPU
BEFORE UPDATE OF Prix
ON PRODUIT
FOR EACH ROW
When (New.Prix > Old.Prix * 1.1 )
BEGIN
        :New.Prix := :Old.Prix * 1.1 ;
END;
/
```

12.1

```
UPDATE PRODUIT
SET Prix = 15.99
WHERE Codprod=10;
```

	Codprod	Prix	....
Ligne avant (OLD)	10	11	

	Codprod	Prix	....
Ligne après (NEW)	10	15.99	

	Codprod	Prix	....
Ligne après (NEW)	10	12.1	

Exemple2: Utilisation d'un TRIGGER pour le maintien d'une contrainte d'intégrité dynamique

Empêcher une augmentation du *PU* d'un *produit* au delà de 10% du prix en cours

```
CREATE OR REPLACE TRIGGER BORNER_AUGMENTPU
BEFORE UPDATE OF Prix ON PRODUIT
FOR EACH ROW
When (New.Prix > Old.Prix * 1.1 )
BEGIN
    RAISE_APPLICATION_ERROR ( -20999, 'Violation de la Contrainte ') ;
END;
/
```

```
UPDATE PRODUIT
SET Prix = 15.99
WHERE Codprod=10;
```

ERROR -20999, 'Violation de la Contrainte

	Codprod	Prix	....
Ligne avant (OLD)	10	11	

	Codprod	Prix	....
Ligne après (NEW)	10	15.99	

Exemple3 : Utilisation d'un TRIGGER pour le maintien d'une contrainte d'intégrité statique

- $0 < codcli < 10000$

```
CREATE OR REPLACE TRIGGER VERIFIERNOCLIENT
BEFORE INSERT OR UPDATE OF Codecli ON CLIENT
FOR EACH ROW
WHEN ( New.Codecli<=0 ) OR ( New.Codecli>=10000 )
BEGIN
    RAISE_APPLICATION_ERROR ( -20009, ' Numéro du client incorrect ' ) ;
END;
```

- **N.B:** CHECK est préférable !

Exemple 4

- **Lors d'un achat, la quantité à commander d'un produit ne peut pas dépasser la quantité en stock disponible**

```
CREATE OR REPLACE TRIGGER VERIFIERSTOCK
BEFORE INSERT OR UPDATE (QteCom) ON Detail
FOR EACH ROW
DECLARE
S Produit.Qte%type;
BEGIN
SELECT Qte INTO S FROM Produit WHERE CodProd=:New.CodProd;
If ( :New.Qtecom> S) Then
    RAISE_APPLICATION_ERROR ( -20009, ' Quantité demandée non disponible') ;
End if;
END;
/
```

Produit (CodProd, Libelle, Qte)
Commande (CodCom, Datcom, ...)
Detail (CodCom, CodProd, QteCom)

Exercices (1/2)

1) Ajouter dans la table Formations une colonne **nbapp** de type entier et valeur par défaut 0: **Alter Table** formations **Add** nbapp number default 0;

2) Ecrire un bloc PL/SQL qui modifie **nbapp** pour chaque formation.

3) - Ecrire un trigger permettant d'incrémenter **nbapp** d'une formation chaque fois qu'on ajoute une inscription pour cette formation.

   - Ecrire une requête qui affiche **idform** et **nbapp** de la table Formations.

   - Ajouter une inscription pour une formation de votre choix.

   - Vérifier si **nbapp** de cette formation a été modifié.

4) - Ecrire un trigger permettant le rejet de l'ajout d'une formation dont **nbapp** est négatif.

   - Vérifiez le fonctionnement de ce trigger en ajoutant une formation dont **nbapp**<0

Exercices (2/2)

5) - Ecrire un trigger permettant le rejet de l'ajout d'un employé (table **Emp**) si son salaire n'appartient pas à un intervalle [**Losal**, **Hisal**] de la table **Salgrade**.

   - Insérer un employé dont le salaire est 600 (il doit être rejeté)

   Insert into emp (empno, ename, sal) values (300, 'Amine', 600);

   - Désactiver ce trigger.

   - Lancer la requête 2). Que remarque-t-on?

6) - Ecrire un trigger permettant le rejet de l'ajout d'un employé si son **MGR** n'existe pas dans la table **Emp**.

   - Insérer un employé dont le MGR est 10. Que remarque-t-on?

Les triggers sur des vues (1)

- Une table possède des **méta-données** et un **contenu**.
- Vue: Une table **virtuelle** dont le **schéma** et le **contenu** sont dérivés de la base réelle par une sélection depuis une ou plusieurs table.
- La définition d'une vue est stockée dans la métabase (dictionnaire de données).
- Une vue **concrète** a un contenu stocké sur disque lors de sa création.
- Une vue est interrogée comme une table.
- Une vue **mono-table** peut être mise à jour (insert, update, delete) ce qui implique automatiquement la mise à jour de la table correspondante.

Les triggers sur des vues (2)

- Création de vue:

CREATE VIEW *nom_vue* [(*LISTE D'ATTRIBUT*)] **AS**
Requête_select
[**WITH CHECK OPTION**]

N.B: La clause **WITH CHECK OPTION** spécifie que les lignes insérés ou mises à jour via la vue doivent satisfaire aux conditions de la clause **WHERE** dans la requête **Select**.

- **Exemple**:

```
Sql> Create view Finance as Select idform, descForm, prix, datdebut, datfin
        From Formation Where descForm like '%finance%' and prix <5000
        With check option;
Sql> desc Finance ;
     …..
Sql> Insert into Finance
     values (23, 'finance islamique', 4500, to_date('12/1/2017','dd/mm/yyyy'),
                                        to_date('15/1/2017','dd/mm/yyyy'));
```

Les triggers sur des vues (3)

```
CREATE VIEW  emp_v As Select empno, ename, sal*1.2 as sal_revu
From Emp Where job='CLERK';
```

```
SQL> desc emp_v
 Nom             NULL ?   Type
 --------------- -------- ------------
 EMPNO           NOT NULL NUMBER(4)
 ENAME                    VARCHAR2(10)
 SAL_REVU                 NUMBER
```

```
SQL> select * from emp_v;

     EMPNO ENAME        SAL_REVU
---------- ---------- ----------
      7369 SMITH             960
      7876 ADAMS            1320
      7900 JAMES            1140
      7934 MILLER           1560
```

```
SQL> select empno, ename, sal, job From Emp where empno=7369;

     EMPNO ENAME             SAL JOB
---------- ---------- ---------- ---------
      7369 SMITH              800 CLERK
```

```
Update emp_v Set ename='Steve'
Where Empno =7369;
```

```
SQL> Update emp_v Set Sal_revu= 4600 Where Empno=7369;
Update emp_v Set Sal_revu= 4600 Where Empno=7369
                *
ERREUR Ó la ligne 1 :
ORA-01733: les colonnes virtuelles ne sont pas autorisÚes ici
```

On peut modifier les colonnes **empno** et **ename** dans la vue **emp_v**. Les modifications seront faites implicitement dans la table **Emp**.

95

Les triggers sur des vues (4)

```
SQL> CREATE  OR  REPLACE   TRIGGER    modif_emp_v
 2     INSTEAD  OF  UPDATE  ON   emp_v
 3     FOR EACH ROW
 4     BEGIN
 5      UPDATE   emp    SET   empno= :New.empno ,
 6           ename= :New.ename ,
 7           sal = :New.sal_revu  -  :New.sal_revu * 0.2
 8     WHERE empno = :New.empno ;
 9   END;
10  /
```

PL/SQL - N.EL FADDOULI

96

Les triggers sur des vues (5)

```
SQL> select * from emp_v where empno=7369;

     EMPNO ENAME        SAL_REVU
---------- ---------- ----------
      7369 SMITH             960
SQL> Update emp_v Set Sal_revu= sal_revu+100 Where Empno=7369;

1 ligne mise Ó jour.
SQL> select empno, ename, sal, job from emp where empno=7369;

     EMPNO ENAME             SAL JOB
---------- ---------- ---------- ---------
      7369 SMITH              848 CLERK
SQL> select * from emp_v where empno=7369;

     EMPNO ENAME        SAL_REVU
---------- ---------- ----------
      7369 SMITH          1017,6
```


Les triggers sur des vues (6)

Pilote			
numP	nomP	ageP	codeC
1	Naciri	35	RM
2	Salmi	30	RM
3	Cristine	50	AF

Compagnie	
codeC	nomC
RM	Royal Air
AF	Air France

```
Create view vue_PC as
Select numP , nomP, p.codeC, nomc
        from Pilote P, Compagnie C
        where C.codeC = P.codeC
        with check Option
```

Insert into vue_PC (numP, nomP, codeC, nomC) values (4, 'Kattani', 'RM', ' Royal Air')

Problème : ORA-01776: cannot modify more than one base table through a join view

Solution: trigger de relais

Les triggers sur des vues (7)

▪ Le déclencheur **INSTEAD OF** permet l'insertion, la modification et la suppression de lignes à travers une vue multitable. **C'est un trigger de relais.**

Exemple:

vue_PC (numP , nomP, codeC, nomC **)**

```
Create or Replace trigger tg_PC INSTEAD OF INSERT ON vue_PC
Begin
 if :New.nomP is NULL or :New.numP is NULL or :New.codeC is null Then
        Raise_Application_Eror ( -21000, ' violation possible de contrainte')
Else
        Insert Into Compagnie values( :New.codeC,:New.nomC)
        Insert Into Pilote values ( :New.numP, :New.nomP, null, :New.codeC)
End if ;
End;
```


Exercices

1. Créer la vue **Statistiques** contenant le montant réalisé par chaque formation (idform, titre, datdebut, datfin, montant réalisé) y compris celles où il n'y a pas d'inscrits.
2. Ecrire une requête pour afficher le contenu de la vue **Statistiques**.
3. Modifier le prix d'une formation
4. Exécuter la requête de la question 2 et comparer le résultat obtenu avec celui obtenu dans 2.
5. Utiliser la vue **Statistiques** pour déterminer le montant maximal réalisé.
6. Utiliser la vue **Statistiques** pour déterminer les formations qui ont réalisé le montant maximal.
7. Insérer une ligne dans la vue **Statistiques**. Que remarque-t-on? Pourquoi?
8. Ecrire un déclencheur de relais pour que la ligne prévue pour insertion dans **Statistiques** soit insérée dans la table **Formation**.

Prédicats de déclenchement

- Lorsqu'on utilise un seul trigger pour plusieurs événements, les fonctions **INSERTING**, **UPDATING** et **DELETING** de type **boolean** permettent d'identifier l'événement déclencheur.
- **Exemple**: Insérer une ligne dans **emp_log** à chaque opération **DML** sur **Emp**

emp_log (emp_no number, who char (12) , operation char(1))

```
CREATE OR REPLACE TRIGGER  emp_log_t
AFTER INSERT OR UPDATE OR DELETE  ON  emp
FOR EACH ROW
DECLARE
   A  CHAR(1);
BEGIN
   IF INSERTING THEN
      A:= 'I';  INSERT INTO emp_log (emp_no, who,  operation) VALUES (:new.empno, USER, A);
   ELSIF UPDATING THEN
      A := 'U'; INSERT INTO emp_log (emp_no, who, operation) VALUES (:new.empno, USER, A);
   END IF;
END;
```

UPDATING ('sal')

Triggers en cascade

- Un trigger est déclenché suite à un événement comme l'exécution d'une instruction SQL (update, delete, insert, alter, ...)
- Un trigger peut lancer à son tour une requête SQL.
- On se retrouve avec des triggers dont un peut lancer un autre.
- Oracle limite ces lancements en cascade à 32, au-delà de cette limite on aura une exception.

Les triggers système

- Les déclencheurs système permettent d'**auditer** le démarrage et l'arrêt du serveur, les erreurs de serveur et les activités de connexion et de déconnexion des utilisateurs.
- Ils sont pratiques pour suivre la durée des connexions par utilisateur, la disponibilité du serveur de base de données, ...etc
- Syntaxe:

CREATE [**OR REPLACE**] **TRIGGER** *trigger_name*
{**BEFORE** | **AFTER**} *database_event* **ON** {**DATABASE** | **SCHEMA**}
[**DECLARE**]
Variables
BEGIN
Instructions
END;

database_event : **Logon**, **Logoff**, **Startup**, **Shutdown**, ….

Le choix de **After** ou **Before** dépend du type d'évènement. Par exemple, **Before** n'est pas possible pour **Logon**.

Les triggers système

- **Exemple 1**: Un trigger d'audit de connexion d'un compte spécifique

Dans le schéma **c##emi**:

```
SQL>  Create Table event_log (compte varchar2(30),
                              event  varchar2(10), event_date   date);
SQL>  Create or Replace Trigger   conn_trg
        After   Logon on  SCHEMA
        begin
                insert into event_log   values (USER, 'Connect',  sysdate);
        end;
        /
SQL>disc
SQL>conn   c##emi/Emi_1234
SQL>Select * from event_log;
```

Trigger déclenché chaque fois que l'utilisateur qui l'a crée se connecte

Les triggers système

- **Exemple 2**: Un trigger d'audit de connexion de tous les utilisateurs

Dans le schéma **c##emi**:

```
SQL>  Create Table event_log (compte varchar2(30),
                              event  varchar2(10), event_date   date);
```

Dans le schéma de l'administrateur **SYS**:

```
SQL>  Create or Replace Trigger   conn_trg
          After   Logon on DATABASE
          begin
          insert into c##emi.event_log   values (USER, 'Connect',  sysdate);
          end;
          /
SQL>disc                    SQL>conn / as sydba
SQL>Select * from event_log
```

Particularités des TRIGGER Oracle

- Pas de SELECT dans le WHEN
- :NEW, :OLD
- Omettre le mot-clé ROW dans REFERENCING
- Corps en PL/SQL
- Syntaxte *:nomColonne*
- Pas de COMMIT/ROLLBACK dans un TRIGGER
  - procédure PL/SQL RAISE_APPLICATION_ERROR
- IF INSERTING, DELETING, UPDATING.
- Événements non standards
  - INSTEAD OF, STARTUP, LOGON, ...
- Problème avec table en mutation (modifée par l'événement déclencheur)
- etc.