

Déclencheurs (triggers)

- Les fonctions pour déclencheurs peuvent être écrites dans la plupart des langages de procédure disponibles:
 - PL/pgSQL-Langage de **procédures SQL**
 - PL/Tcl-Langage de **procédures Tcl**
 - PL/Perl-langage de **procédure Perl**
 - PL/Python-langage de **procédures Python**
- Consulter votre langage de procédures pour découvrir les spécificités de l'écriture des déclencheurs dans ce langage
- Il est possible d'écrire une fonction déclencheur en C

Déclencheurs (triggers)

- Un déclencheur spécifie que la base de données doit exécuter automatiquement une fonction donnée chaque fois qu'un certain type d'opération est exécutée.
- Les fonctions triggers peuvent être définies pour s'exécuter avant ou après une commande **INSERT**, **UPDATE** ou **DEELETE** soit une fois par ligne modifiée soit une fois par instruction SQL
- Si un évènement déclencheur se produit, le gestionnaire de déclencheurs est appelé au bon moment pour gérer l'événement
- La fonction déclencheur doit être définie avant que le déclencheur lui-même ne puisse être créé

Déclencheurs (triggers)

- La fonction déclencheur doit être déclarée comme une fonction ne prenant aucun argument et retournant un type **trigger**
- La fonction déclencheur reçoit ses entrées via une structure **triggerData** passée spécifiquement, et non pas sous la forme d'arguments ordinaires de fonctions
- Une fois qu'une fonction déclencheur est créée, le déclencheur (trigger) est créé avec la fonction **CREATE TRIGGER**
- La même fonction déclencheur est utilisable par plusieurs déclencheurs

Comment créer un Déclencheur avec SQL?

Synopsis

```
CREATE [ CONSTRAINT ] TRIGGER nom { BEFORE | AFTER | INSTEAD OF } { événement [ OR ... ] }  
ON nom_table  
[ FROM nom_table_referencee ]  
[ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]  
[ REFERENCING { { OLD | NEW } TABLE [ AS ] nom_relation_transition } [ ... ] ]  
[ FOR [ EACH ] { ROW | STATEMENT } ]  
[ WHEN ( condition ) ]  
EXECUTE { FUNCTION | PROCEDURE } nom_fonction ( arguments )
```

où *événement* fait partie de :

```
INSERT  
UPDATE [ OF nom_colonne [, ... ] ]  
DELETE  
TRUNCATE
```

Version simplifiée

```
CREATE TRIGGER trigger_name  
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}  
ON table_name FOR EACH ROW  
trigger_body;
```

Il convient de décrire chacun de ces éléments :

- **CREATE TRIGGER trigger_name** : c'est l'envoi de la requête TRIGGER. Il s'agit alors de nommer votre déclencheur.
- **{BEFORE | AFTER}** : cela concerne la temporalité du déclencheur. Se déroulera-t-il avant ou après que l'événement n'apparaisse ?
- **{INSERT | UPDATE | DELETE}** : il s'agit de l'événement. En l'occurrence, ajouter, mettre à jour ou supprimer. Bien évidemment, il est possible de prévoir n'importe quel type d'événement.
- **ON table_name** : c'est la base de données concernée par la requête TRIGGER SQL.
- **FOR EACH ROW** : il s'agit de préciser les lignes affectées par le déclencheur.
- **trigger_body** : c'est la description de votre déclencheur. Que se passera-t-il si l'événement survient ?

Exemple de requête SQL créant un trigger

- La requête SQL ci-dessous est un exemple concret de requête SQL pour créer un trigger (déclencheur) :

```
CREATE OR REPLACE TRIGGER trigg_example
BEFORE INSERT OR UPDATE ON table_example
FOR EACH ROW
WHEN (new.no_line > 0)
DECLARE
    evol_exemple number;
BEGIN
    evol_exemple := :new.exemple - :old.exemple;
    DBMS_OUTPUT.PUT_LINE('  evolution : ' || evol_exemple);
END;
```

Déclencheur mode ligne/instruction

- **mode ligne**, la fonction du déclencheur est appelée une fois pour chaque ligne affectée par l'instruction qui a lancé le déclencheur
- un déclencheur **mode instruction** n'est appelé qu'une seule fois lorsqu'une instruction appropriée est exécutée, quelque soit le nombre de lignes affectées par cette instruction
- En particulier, une instruction n'affectant aucune ligne résultera toujours en l'exécution de tout déclencheur mode instruction applicable
- Ces deux types sont quelque fois appelés respectivement des *déclencheurs niveau ligne* et des *déclencheurs niveau instruction*.

Déclencheur avant ou après

- Les **déclencheurs avant au niveau instruction** se déclenchent naturellement avant que l'instruction ne fasse quoi que ce soit alors que **les déclencheurs après au niveau instruction** sont exécutés à la fin de l'instruction.
- Les **déclencheurs avant au niveau ligne** s'exécutent immédiatement avant qu'une ligne particulière ne soit traitée alors que les **déclencheurs après au niveau ligne** s'exécutent à la fin de l'instruction (mais avant tout déclencheur après au niveau instruction).

Exemple 1 de déclencheur

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

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

```
CREATE OR REPLACE TRIGGER BORNER_AUGMENT PU
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

	Codprod	Prix
Ligne avant (OLD)	10	11	
Ligne après (NEW)	10	15.99	

	Codprod	Prix
Ligne après (NEW)	10	12.1	

Exemple 2 de déclencheur

- 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_AUGMENT PU
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	
Ligne après (NEW)	10	15.99	

Exemple 3 de déclencheur

- Utilisation d'un TRIGGER pour le maintien d'une contrainte d'intégrité statique
- $0 < \text{codcli} < 10000$

```
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;
```

Exemple 4 de déclencheur

- 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)

Exemple 5 de déclencheur

- Exécutez la fonction `check_account_update` quand une ligne de la table `accounts` est sur le point d'être mise à jour :

```
CREATE TRIGGER check_update  
BEFORE UPDATE  
ON accounts  
FOR EACH ROW EXECUTE PROCEDURE check_account_update();
```

- Idem, mais avec une exécution de la fonction seulement si la colonne `balance` est spécifiée comme cible de la commande `UPDATE` :

```
CREATE TRIGGER check_update  
BEFORE UPDATE OF balance  
ON accounts  
FOR EACH ROW EXECUTE PROCEDURE check_account_update();
```

Exemple 6 de déclencheur

- Cette forme exécute la fonction seulement si la colonne balance a réellement changé de valeur :

```
CREATE TRIGGER check_update
BEFORE UPDATE
ON accounts
FOR EACH ROW WHEN (OLD.balance IS DISTINCT FROM NEW.balance)
EXECUTE PROCEDURE check_account_update();
```

- Appelle une fonction pour tracer les mises à jour de la table accounts, mais seulement si quelque chose a changé :

```
CREATE TRIGGER log_update
AFTER UPDATE ON accounts
FOR EACH ROW WHEN (OLD.* IS DISTINCT FROM NEW.*)
EXECUTE PROCEDURE log_account_update();
```

Exemple 7 de déclencheur

- Exécute la fonction `view_insert_row` pour chacune des lignes à insérer dans la table sous-jacente à la vue `my_view` :

```
CREATE TRIGGER view_insert  
  INSTEAD OF INSERT ON my_view  
  FOR EACH ROW  
  EXECUTE PROCEDURE view_insert_row();
```

- Exécute la fonction `check_transfer_balances_to_zero` pour chaque commande pour confirmer que les lignes de transfert engendrent un net de zéro :

```
CREATE TRIGGER transfer_insert  
  AFTER INSERT ON transfer  
  REFERENCING NEW TABLE AS inserted  
  FOR EACH STATEMENT EXECUTE PROCEDURE check_transfer_balances_to_zero();
```

Exemple 8 de déclencheur

- Exécute la fonction `check_matching_pairs` pour chaque ligne pour confirmer que les changements sont faits sur des paires correspondantes au même moment (par la même commande) :

```
CREATE TRIGGER paired_items_update  
AFTER UPDATE ON paired_items  
REFERENCING NEW TABLE AS newtab OLD TABLE AS oldtab  
FOR EACH ROW  
EXECUTE PROCEDURE check_matching_pairs();
```


Exemple 9 de déclencheur

```
CREATE OR REPLACE TRIGGER MonTrigger
  BEFORE INSERT OR DELETE for (att1) OR UPDATE ON maTable
  FOR EACH ROW

  BEGIN

    IF INSERTING OR UPDATING THEN

      ELSE IF DELETING
        IF maCondition THEN
          .....
        END IF;
      END IF;

    IF DELETING THEN
      .....
    END IF;

  END;
```

Que retourne les Déclencheurs?

- Les fonctions déclencheurs appelées par des déclencheurs niveau instruction devraient toujours renvoyer **NULL**.
- Les fonctions déclencheurs appelées par des déclencheurs niveau ligne peuvent renvoyer une ligne de la table vers l'exécuteur appelant, s'ils le veulent.
- Un déclencheur niveau ligne exécuté avant une opération a les choix suivants :
 - Il peut retourner un pointeur NULL pour sauter l'opération pour la ligne courante. Ceci donne comme instruction à l'exécuteur de ne pas exécuter l'opération niveau ligne qui a lancé le déclencheur (l'insertion ou la modification d'une ligne particulière de la table).
 - Pour les déclencheurs **INSERT** et **UPDATE** de niveau ligne uniquement, la valeur de retour devient la ligne qui sera insérée ou remplacera la ligne en cours de mise à jour. Ceci permet à la fonction déclencheur de modifier la ligne en cours d'insertion ou de mise à jour.

Que retourne les Déclencheurs?

- Un déclencheur avant niveau ligne qui ne serait pas conçu pour avoir l'un de ces comportements doit prendre garde à retourner la même ligne que celle qui lui a été passée comme nouvelle ligne
- c'est-à-dire : pour des déclencheurs **INSERT** et **UPDATE**: la nouvelle (new) ligne et pour les déclencheurs **DELETE**: l'ancienne (old) ligne
- La valeur de retour est ignorée pour les déclencheurs niveau ligne lancés après une opération. Ils peuvent donc renvoyer la valeur NULL

Un évènement et plusieurs Déclencheurs

- Si plus d'un déclencheur est défini pour le même événement sur la même relation, les déclencheurs seront **lancés dans l'ordre alphabétique de leur nom.**
- Dans le cas de déclencheurs avant, **la ligne renvoyée par chaque déclencheur, qui a éventuellement été modifiée, devient l'argument du prochain déclencheur.**
- Si un des déclencheurs avant renvoie un pointeur NULL, **l'opération est abandonnée pour cette ligne et les déclencheurs suivants ne sont pas lancés.**

Utilisation des Déclencheurs

- Les déclencheurs avant en mode ligne sont typiquement utilisés **pour vérifier ou modifier les données qui seront insérées ou mises à jour.**
- Par exemple, un déclencheur avant pourrait être utilisé pour *insérer l'heure actuelle dans une colonne de type timestamp ou pour vérifier que deux éléments d'une ligne sont cohérents.*
- Les déclencheurs après en mode ligne sont pour la plupart utilisés **pour propager des mises à jour vers d'autres tables ou pour réaliser des tests de cohérence avec d'autres tables.**

Déclencheur en cascade

- Si une fonction déclencheur exécute des commandes SQL, alors ces commandes peuvent lancer à leur tour des déclencheurs.
- On appelle ceci un déclencheur en cascade. Il n'y a pas de limitation directe du nombre de niveaux de cascade.
- Il est possible que les cascades causent un appel récursif du même déclencheur

Arguments des Déclencheurs

- Quand un déclencheur est défini, **des arguments peuvent être spécifiés pour lui.**
- Par exemple, il pourrait y avoir une fonction déclencheur généralisée qui prend comme arguments deux noms de colonnes et place l'utilisateur courant dans l'une et un horodatage dans l'autre.
- Correctement écrit, cette fonction déclencheur serait indépendante de la table particulière sur laquelle il se déclenche.

Maintenance des Déclencheurs (1)

- **Création/suppression d'un trigger**
 - CREATE TRIGGER nom_déclencheur;
 - REPLACE TRIGGER nom_déclencheur;
 - DROP TRIGGER nom_déclencheur;
- **Activation/désactivation d'un trigger**
 - ALTER TRIGGER nom_déclencheur DISABLE;
 - ALTER TRIGGER nom_déclencheur ENABLE;
- **Activer/désactiver tous les triggers définis sur une table**
 - ALTER TABLE nom_table DISABLE ALL TRIGGERS;
 - ALTER TABLE nom_table ENABLE ALL TRIGGERS

Maintenance des Déclencheurs (2)

- Les informations sur les triggers sont visibles à travers les vues du dictionnaire de données
 - **USER_TRIGGERS** pour les triggers appartenant au schéma
 - **ALL_TRIGGERS** pour les triggers appartenant aux schémas accessibles
 - **DBA_TRIGGERS** pour les triggers appartenant à tous les schémas

Condition d'erreurs

- Dans un trigger, différentes conditions d'erreur peuvent être définies à travers la procédure **RAISE_APPLICATION_ERROR**
- Le numéro d'erreur peut varier de 20001 à 20999
- Si une erreur est levée, **l'événement du trigger ne se réalise pas**

Appel de procedures

- A partir d'un trigger de procédures peuvent être appelées

```
create or replace trigger BOOKSHELF_AFT_INS_ROW
after insert on BOOKSHELF_AUDIT
for each row
begin
call INSERT_BOOKSHELF_AUDIT_DUP(:new.Title, :new.Publisher,
    :new.CategoryName, :new.Old_Rating, :new.New_Rating,
    :new.Audit_Date);
end;
/
```

Triggers Vs Contraintes d'intégrité

- Possibilité d'utilisation dans les mêmes situations
- Utiliser les triggers lorsque l'utilisation de contraintes n'est pas possible
- Lorsqu'il est possible utiliser les contraintes
 - NOT NULL, UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK
 - DELETE CASCADE
 - DELETE SET NULL

Triggers en CASCADE

SQL Statement
UPDATE t1 SET ...;

Fires the
UPDATE_T1
Trigger

UPDATE T1 Trigger
BEFORE UPDATE ON t1
FOR EACH ROW
BEGIN
.
.
INSERT INTO t2 VALUES (...);
.
.
END;

Fires the
INSERT_T2
Trigger

INSERT T2 Trigger
BEFORE INSERT ON t2
FOR EACH ROW
BEGIN
.
.
INSERT INTO ... VALUES (...);
.
.
END;

etc.

- Faire attention au déclenchement en cascade des triggers

Ordre dans le déclenchement

- Par défaut l'ordre suivant :

1. All BEFORE statement triggers
2. All BEFORE row triggers
3. All AFTER row triggers
4. All AFTER statement triggers

- Clause FOLLOWS pour triggers du même type :

```
CREATE OR REPLACE TRIGGER <trigger_name>  
[FOLLOWS | PRECEDES <schema.trigger_name>]
```

Langage de programmation et Déclencheurs

- Chaque langage de programmation supportant les déclencheurs a sa propre méthode pour rendre les données en entrée disponible à la fonction du déclencheur.
- Cette donnée en entrée inclut le type d'événement du déclencheur (c'est-à-dire **INSERT** ou **UPDATE**) ainsi que tous les arguments listés dans **CREATE TRIGGER**.
- Pour un déclencheur niveau ligne, la donnée en ligne inclut aussi la ligne **NEW** pour les déclencheurs **INSERT** et **UPDATE** et/ou la ligne **OLD** pour les déclencheurs **UPDATE** et **DELETE**.
- Les déclencheurs niveau instruction n'ont actuellement aucun moyen pour examiner le(s) ligne(s) individuelle(s) modifiées par l'instruction.