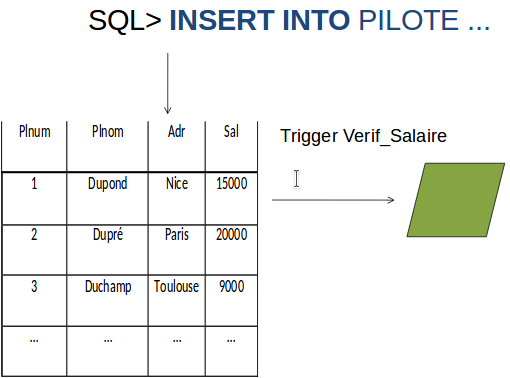
Les Triggers

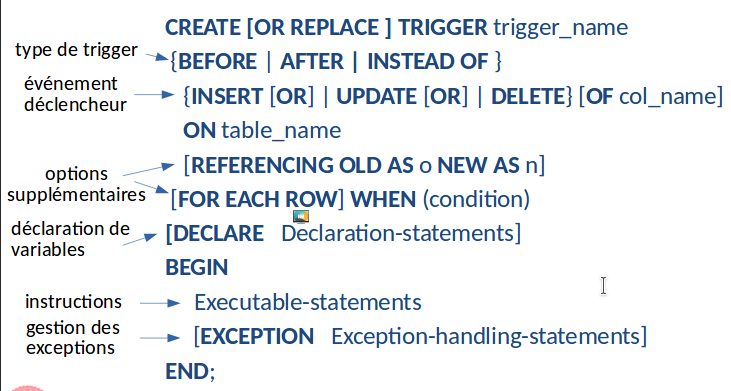
Présentation :

Un déclencheur est un traitement (sous forme de bloc PL/SQL) qui s’exécute automatiquement en réponse à un événement. Deux types :

* Déclencheur base de données
* Déclencheur d’application (Rappel : les contraintes applicatives qui ont été définies lors de l’analyse de l’application)



Syntaxe d’un trigger :



Règle de nommage du trigger :

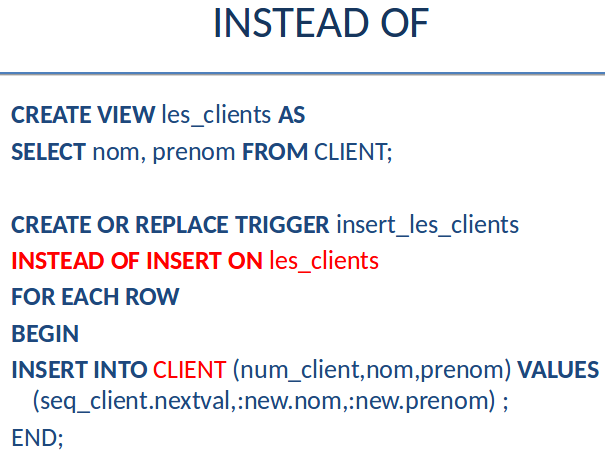
Le nom d’un trigger doit être unique dans un même schéma. Même s’il peut avoir le même nom qu’un autre objet (table, vue, procédure) il est préférable d’éviter pour ne pas avoir de conflits.

IL existe 3 types de triggers précisant le moment de son exécution :

* BEFORE : exécuté avant l’ordre LMD qui l’a déclenché
* AFTER : après
* INSTEAD OF : exécuté à la place de l'exécution de l’ordre LMD qui l’a déclenché

Élément d’un trigger :

1. Événement :

Indique quel ordre SQL déclenche le traitement (INSERT, UPDATE, DELETE, ou toute combinaison de ces ordres). Pour UPDATE, on peut avoir une liste de colonnes, le trigger ne se déclenche que si l’instruction UPDATE porte sur l’une au moins des colonnes précisées dans la liste. S’il n’y a pas de liste, le trigger est déclenché pour toute instruction UPDATE portant sur la table.

1. Table :

La définition précise la table associée au trigger. Une et une seule table. Pas de vue (voir INSTEAD OF)

1. le type d’un trigger détermine :

Quand SGBD déclenche le trigger → défini pas BEFORE, AFTER, INSTEAD OF

Combien de fois le traitement doit s’exécuter suite à l'événement qui l’a déclenché → défini par FOR EACH ROW

Les 2 types de triggers :

Oracle propose 2 types de triggers :

* Les triggers lignes qui se déclenchent individuellement pour chaque ligne de la table affectée par le trigger. Pour spécifier un trigger ligne : FOR EACH ROW
* Les trigger globaux qui ne se déclenchent qu’une fois (option par défaut) - début ou fin de transaction.

Pour chaque type, il existe 3 possibilités précisent le moment de son exécution : BEFORE, AFTER, INSTEAD OF

Remarque : Les triggers AFTER sur ligne sont plus efficaces que les triggers BEFORE sur

ligne car ils ne nécessitent qu’une seule lecture des données . Le INSTEAD OF est utilisé

souvent pour faire des mises à jour via des VUES.

Restriction Triggers en ligne

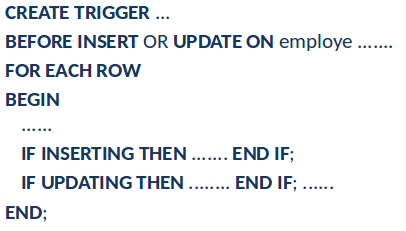
Il est possible d’ajouter une restriction sur les lignes via une expression logique SQL : c’est la clause WHEN:

* Cette expression est évaluée pour chaque ligne affectée par le trigger
* Le trigger n’est déclenché sur une ligne que si l’expression WHEN est vérifiée pour cette ligne
* L’expression logique ne peut pas contenir une sous requête

Élément d’un déclencheur :

Traitement - corps du déclencheur : Quelles actions à exécuter ?

* Le corps du déclencheur est défini sous forme d’un bloc PL/SQL anonyme
* Il peut contenir du SQL et du PL/SQL
* Il est exécuté si l’instruction de déclenchement se produit et si la clause de restriction WHEN, le cas échéant, est évaluée à vrai.
* Les corps d’un trigger ligne et d’un trigger global sont différents



Les prédicats conditionnels

Quand un trigger comporte plusieurs instructions de déclenchement (INSERT or DELETE or UPDATE), on peut utiliser des prédicats conditionnels (INSTERING, DELETING et UPDATING) pour exécuter des blocs de code spécifiques pour chaque instruction de déclenchement

Noms de corrélation

Il est possible dans un trigger en ligne d’accéder à la nouvelle valeur et à l’ancienne valeur (noms de corrélation), mais pas d'accès à ces valeurs avec un trigger global. Attention : avec INSERT, seule la nouvelle valeur a un sens, avec DELETE, seule l’ancienne a un sens.

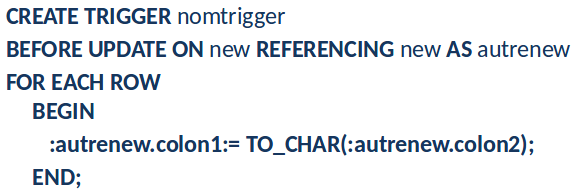
Dans le corps du déclencheur :

* la nouvelle valeur est → :new.nom\_colonne
* l’ancienne valeur est → :old.nom\_colonne

Exemple :

* IF : new.salaire > :old.salaire THEN …

Attention : dans la condition WHEN, on peut aussi consulter ces valeurs, mais il n’y a pas “ : “ avant “ new “, “ old ”

Exemple : WHEN (new.empno>0)

Referencing

Si une table s’appelle NEW ou OLD, il est possible d’utiliser REFERENCING pour éviter l’ambiguïté entre le nom de la table et le nom de corrélation

Exceptions

Si une erreur se produit pendant l’exécution d’un trigger, toutes les mises à jour produites par le trigger ainsi que par l’instruction qui l’a déclenché sont défaites. Possibilité de mettre des opérations dans exception dans un bloc PL/SQL

Raise\_application\_error

Procédure spécifique :

raise\_application\_error(error\_number,error\_message)

* error\_number doit être un entier compris entre - 20 000 et - 20 999
* error\_message doit être une chaîne de 500 caractères maximum
* Quand cette procédure est appelée, elle termine le trigger, défait la transaction (ROLLBACK), renvoie un numéro d’erreur défini par l’utilisateur et un message à l’application

Nombre de triggers par table

On peut avoir au maximum un trigger de chacun des types suivant pour chaque table :

BEFORE UPDATE row / BEFORE DELETE row / BEFORE INSERT statement / BEFORE INSERT row / BEFORE UPDATE statement / BEFORE DELETE statement / AFTER UPDATE row / AFTER DELETE row / AFTER INSERT statement / AFTER INSERT row / AFTER UPDATE statement / AFTER DELETE statement

Mais attention : Il ne peut y avoir qu’un UPDATE même si on change les noms de colonnes

Instruction SQL autorisées

Dans un triggers, les instructions SQL autorisées sont : les instructions du LMD. Celles interdites sont : les instructions du LDD et les instructions de contrôle des transactions (ROLLBACK, COMMIT)

Activation d’un trigger

Un trigger est activé par défaut.

Désactivation d’un trigger : ALTER TRIGGER nomtrigger DISABLE ;

Pour désactiver tous les triggers associés à une table : ALTER TABLE nomtable DISABLE ALL TRIGGERS;

Pour activer un trigger : ALTER TRIGGER nomtrigger ENABLE;

pour activer tous les triggers associés à une table : ALTER TABLE nomtable ENABLE ALL TRIGGERS;

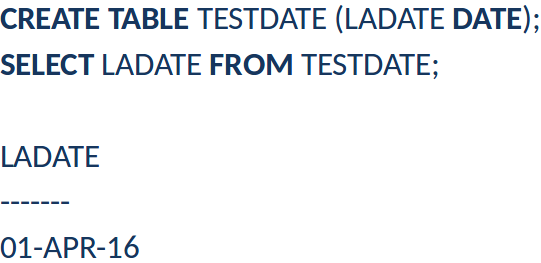
Modification et métabase

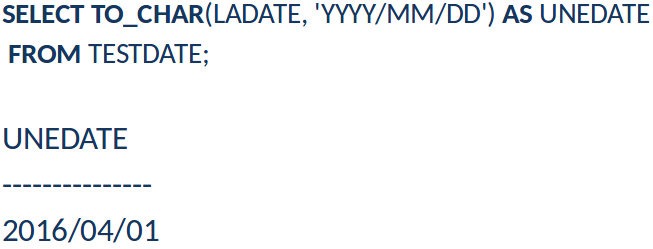
Modification d’un trigger : CREATE OR REPLACE …

ou bien : DROP TRIGGER nom\_trigger

Métabase : tables USER\_TRIGGERS, ALL\_TRIGGERS et DBA\_TRIGGERS

**DATE ET SYSDATE**

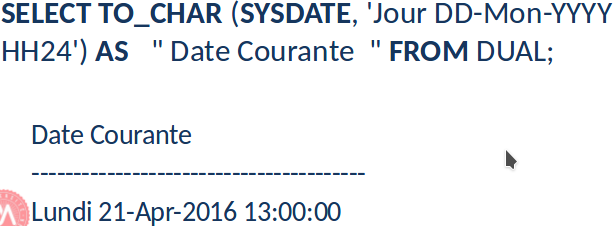


TO CHAR permet de convertir les dates 

TO\_CHAR(<date>,’<format>’), où format :

* MM mois en numérique (04)
* MON nom du mois en abrégé (APR)
* MONTH Nom du mois en entier (APRIL)
* DD Jour du mois (1)
* DY Nom abrégé du jour (FRI)
* YYYY 4-digit de l’année (2016)
* YY 2-digits de l’année (16)
* RR Comme YY, mais les deux digits sont arrondis à l’année dans l’intervalle 1950 à 2049. Ainsi 16 est considéré comme 2016 au lieu de 1016.
* AM (or PM)Indicateur du méridien
* HH Heure du jour (1-12)
* HH24 Heure du jour (0-23)
* MI Minute (0-59)
* SS Seconde (0-59)

Opération inverse : conversion d’une chaîne en format DATE : To\_date (chaine, ‘<format>’) (format même que TO\_CHAR)

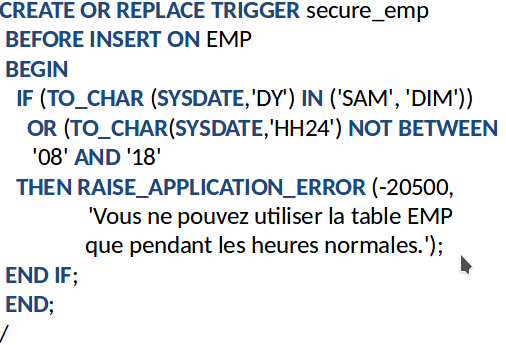
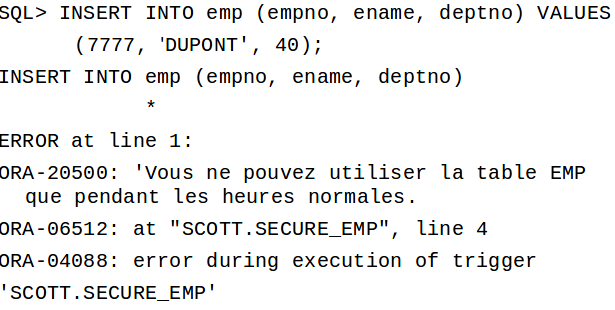
INSERT INTO TESTDATE VALUES (TO DATE(‘2016/APR/02’,’yyyy,mm,dd’); 

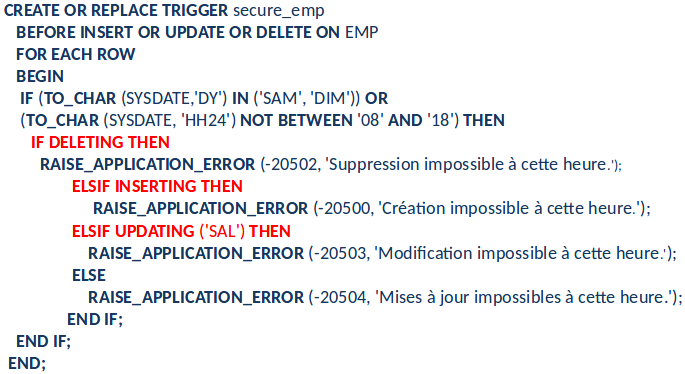
SYSDATE permet de connaître la date système. Peut être utilisé directement dans les triggers. Par contre pour afficher la date système il faut utiliser une relation DUAL.

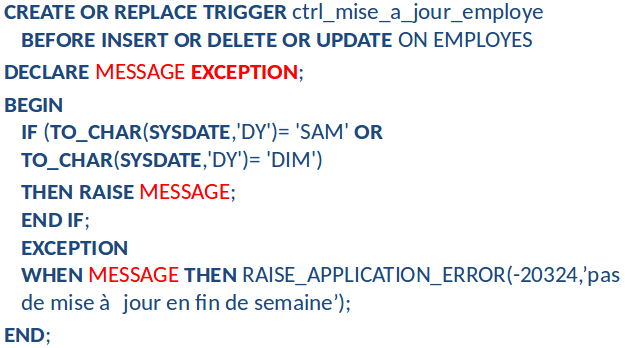
**Exemples de triggers**

Utilisation des triggers

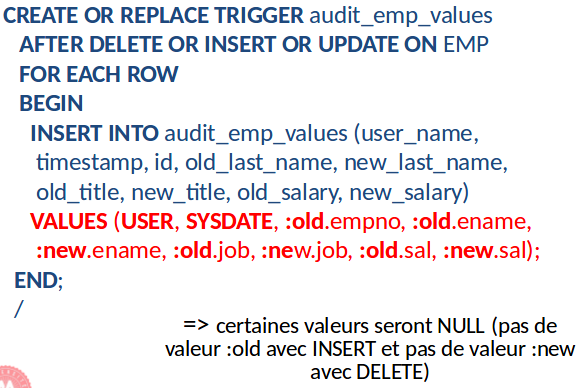
Sécurité, audit, intégrité des données, intégrité référentielle, réplication de données, données dérivées, génération d’événements.

Exemple 1 : sécurité

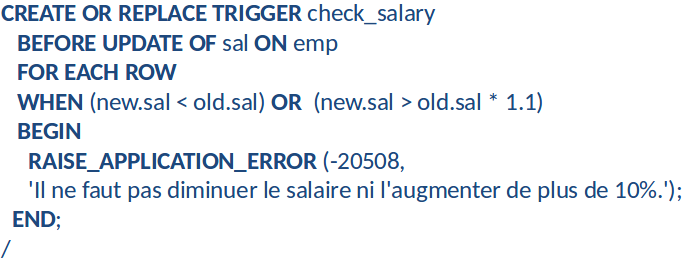
Exemple 2 : sécurité 

Exemple 3 : sécurité 

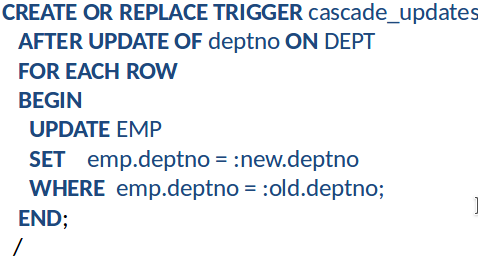
Exemple 4 : audit

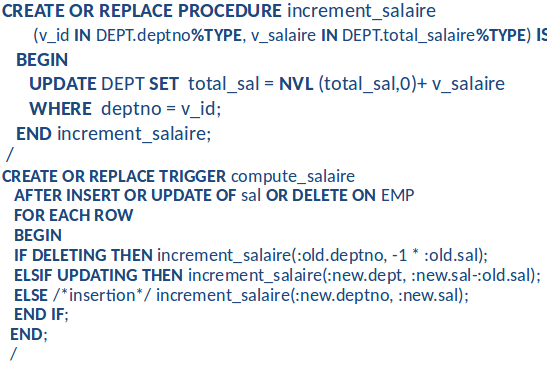


Exemple 5 :vérification intégrité de données



Exemple 6 : vérification intégrité référentielle



Exemple 7 : données dérivées 

Tables en mutation et triggers

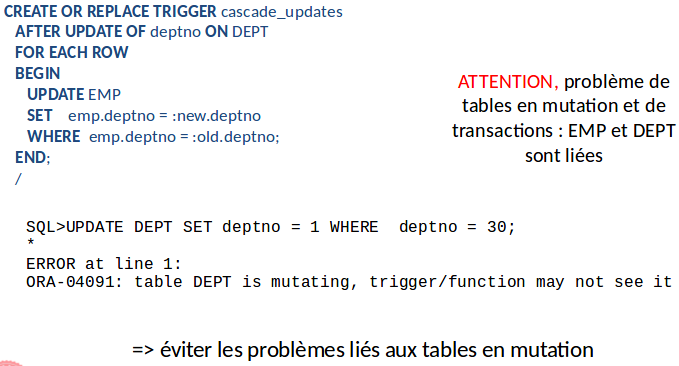
Problème : impossible de déterminer certaines valeurs lors de l'exécution d’une séquence d’opérations appartenant à une même transaction.

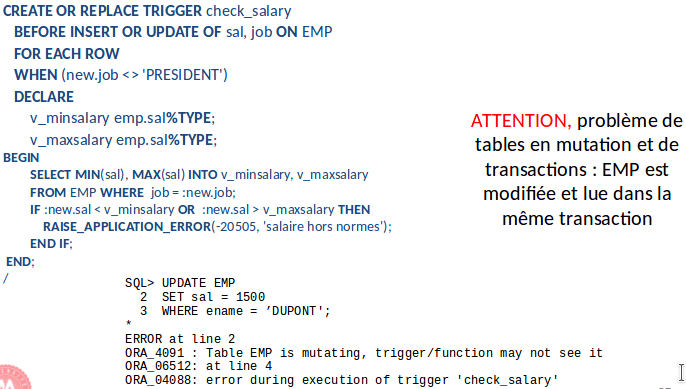
Tables en mutation : table qui a la possibilité de changer →tables qui contient des lignes qui changent de valeur après certaines opérations et qui sont réutilisées avant la validation de la transaction en cours.

Eviter les tables en mutation :

* ne doit pas contenir de lignes qui sont contraintes par des lignes d’autres tables changeantes.
* ne doit pas contenir de lignes qui sont mises à jour et lues en une seule et même opération
* ne doit pas contenir de lignes qui sont mises à jour et lues via d’autres opérations au cours de la même transaction.

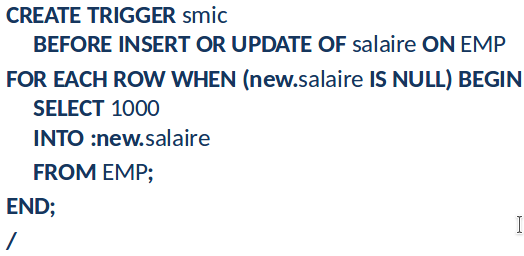
Exemple 8 : table en mutation



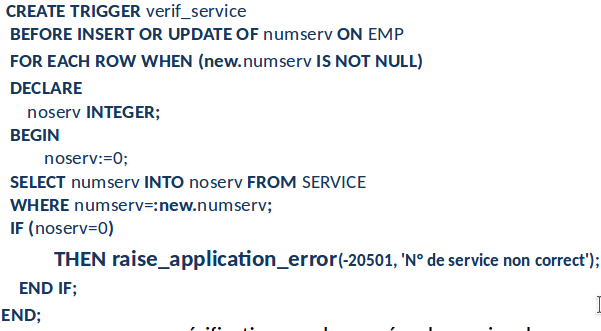
Exemple 9 : table en mutation : 

Règles de bonnes pratiques :

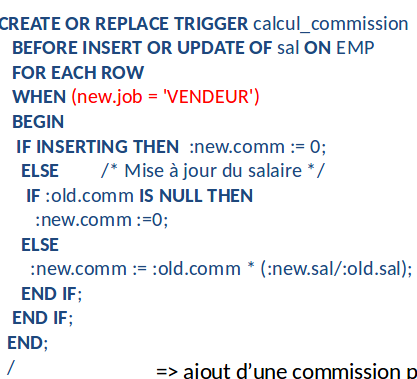
1. Ne pas modifier les données dans des colonnes de clé primaire, clé étrangère ou clé unique d’une table
2. Ne pas mettre à jour les tuples d’une table que vous lisez au cours de la même transaction
3. Ne pas faire d’agrégation sur la table que vous mettez à jour
4. Ne pas lire les données d’une table qui est mise à jour au cours de la même transaction
5. N’utilisez pas d'instructions SQL DCL (Data Control Language) dans les triggers



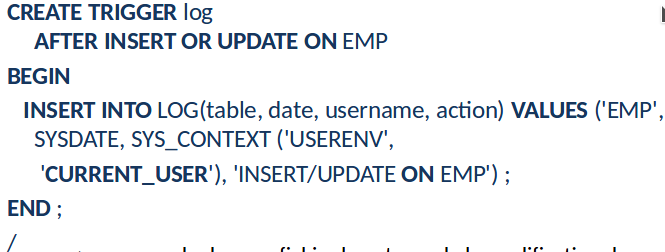
Exemple 10 : ajouter une valeur de 1000 euros lorsque l’employé n’a pas de salaire



Exemple 11 : vérification que le numéro du service de l’employé existe bien



Exemple 12 : ajout d’une commission pour les vendeurs uniquements



Exemple 13 : sauvegarde dans un fichier log : trace de la modification de la table Emp\_tab (moment + utilisateur). N’est exécuté qu’une fois par modification de la table Emp. SYS\_CONTEXT renvoie la valeur du paramètre associé à l’espace de noms de contexte.

Exemple 14 : pour chaque modification (lignes mises à jour), le trigger va calculer puis afficher respectivement l’ancien salaire, le nouveau salaire et la différence entre ces deux salaires.

