TP N°03 (semaines 41-42) : 4h
Les déclencheurs PL/SQL LMD

Rappel : syntaxe générale des déclencheurs LMD (cf support de cours)

```
CREATE [OR REPLACE] TRIGGER nom_décl
Séquence
événement [OR événement]
ON nom_table
[FOR EACH ROW]
[WHEN condition]
[DECLARE declarations]
BEGIN
    Instructions;
END;
```

Séquence : BEFORE ou AFTER
événement : INSERT ou UPDATE ou DELETE
nom_table : table à laquelle le déclencheur est lié
bloc PL/SQL : traitement à réaliser
condition : Si vraie, traitement exécuté

FOR EACH ROW et WHEN : uniquement pour décl. LIGNE

Exercice 1: Mon Premier Trigger (**Déclencheur LMD - Déclencheur d'état**)

a) Lister les agents d'entretien de l'hôtel **encore en poste** (nom-prénom, âge), triés par âge décroissant.

b) Ecrire en PL/SQL un déclencheur d'état **TR_jourOk** permettant de rejeter la mise à jour (insertion – modification – suppression) de la table AGENT_ENTRETIEN **le lundi** (ou un autre jour pour les tests).

Pour connaître le nom du jour d'aujourd'hui, on utilisera la fonction TO_CHAR :

TO_CHAR (SYSDATE,'fmday','NLS_DATE_LANGUAGE = French') renvoie le nom du jour d'aujourd'hui en toutes lettres, en minuscules (day), en français et sans espace inutiles (fm).

La procédure **RAISE_APPLICATION_ERROR**(num_err, msg_err) permet de déclencher une erreur utilisateur (avec num_err compris entre -20001 et -20999) et empêche ainsi la commande déclenchante (par ex. la commande INSERT) d'aboutir.

Compiler le déclencheur, corriger les éventuelles erreurs de syntaxe puis vérifier qu'il apparaît dans la liste de vos déclencheurs.

Dans un script contenant plusieurs requêtes et/ou blocs PL/SQL, il faut terminer le déclencheur par un **/**

Utiliser  pour compiler votre déclencheur.

Les erreurs de compilation apparaissent dans le journal qui s'affiche. Utiliser **CREATE OR REPLACE TRIGGER** pour pouvoir recompiler le trigger après correction des erreurs.

c) Tester le déclencheur en tentant une insertion dans la table **AGENT_ENTRETIEN**

AGT_ID	AGT_NOM	AGT_PRENOM	AGT_DNAIS	AGT_EMB	AGT_SALAIRE
A07	DELON	Alain	01-01-1952	03-03-2016	1500

L'insertion doit provoquer une erreur →

ORA-20120: Désolé pas de modifications de la table AGENT_ENTRETIEN aujourd'hui !

d) Désactiver le déclencheur **jour_ok (ALTER TRIGGER … DISABLE).** Relancer l'insertion.

Exercice 2 : Déclencheur ou contraintes ? (Déclencheur LMD - Déclencheur ligne)

1. Afficher le détail des lignes des factures payées par chèque (PMT_CODE = 'CHQ') et liées à une adresse de l'Aisne (dont le code postal qui commence par 02).
   (Aide : 6 lignes de factures en tout sont attendues pour les factures 1476 et 2500)

   Afficher :

Fac_id	Cli_id	Adr_id	Mnt	Pmt_code

2. Tenter de supprimer la facture N°1476.
   a. Pourquoi cette commande déclenche-t-elle une erreur ?
   b. Quelles solutions pourraient permettre de supprimer automatiquement les lignes des factures lors de la suppression d'une facture ? Laquelle est préférable ?
3. Avant leur suppression, on désire archiver les factures et les lignes de facture. Pour cela :
   a. Créer les tables **OLD_FACT** et **OLD_LG** avec les mêmes champs que les tables **FACTURE** et **LIGNE_FACTURE** (mais les tables doivent être vides, inutile d'ajouter PK et FK).
   (Astuce : créer les tables à partir d'un)
   b. Créer le **déclencheur ligne** (FOR EACH ROW) **TR_Facture** qui supprime (DELETE) les lignes des factures correspondantes (dans la table LIGNE_FACTURE) lorsqu'une facture est supprimée (DELETE ON facture).

      Tester en supprimant la facture N°1476.

      Vérifier que les lignes de la facture ont été supprimées également (relancer la requête du 1). Annuler les suppressions effectuées (ROLLBACK).

Le premier déclencheur (exo 1) était un **déclencheur d'état** (ou déclencheur par ordre) qui ne se déclenche qu'une seule fois quel que soit le nombre de lignes traitées par l'ordre LMD déclencheur.

Un **déclencheur ligne** lui, est exécuté pour chaque ligne affectée par l'ordre DML. Il contient la directive **FOR EACH ROW.**

Dans un déclencheur ligne, **les colonnes de la ligne en cours de modification** sont accessibles par l'intermédiaire des 2 variables de type enregistrement **OLD** et **NEW.**
NEW représente la nouvelle valeur et **OLD** représente la valeur avant modification. Dans la section exécutable (entre BEGIN et END), elles doivent être préfixées comme des variables hôtes avec le symbole **:** (par exemple :new.monChamp ou :old.monChamp)

   c. Modifier le trigger pour qu'il insère avant chaque suppression les lignes correspondantes dans les tables OLD_FACT et OLD_LG. Vérifier.

Exercice 3 : Trigger 3 - Déclencheur ligne FOR EACH ROW

1. Créer un déclencheur **TR_planning** qui vérifie lors de l'enregistrement d'une ligne dans la table PLANNING (insertion ou modification) si la date **PLN_JOUR** est inférieure à la date d'aujourd'hui (utiliser IF) ou non renseignée (NULL) et, si c'est le cas, affecte la date système à PLN_JOUR tronquée au jour : TRUNC(SYSDATE).
   Tester les insertions suivantes puis vérifier en affichant les réservations du client 100 triées par date décroissante (les insertions doivent avoir été insérées à la date du jour).

```
INSERT INTO planning VALUES (1, TO_DATE('01/01/2020','DD/MM/YYYY'),100, 2);
INSERT INTO planning VALUES (2, NULL,100, 2);
```

2. Compléter le déclencheur **TR_Planning** pour qu'il vérifie également lors de l'enregistrement d'une ligne dans la table PLANNING (insertion ou modification) que le nombre de personnes (**nb_pers**) est bien inférieur ou égal au nombre de couchages de la chambre (**chb_couchage**).
   - Si le nombre de personnes est valide, la réservation est acceptée.
     Afficher : « `Réservation Enregistrée` ».
   - Si le nombre de personnes n'est pas valide, la réservation n'est pas insérée.
     Afficher alors un message d'erreur.

La procédure **RAISE_APPLICATION_ERROR** (num_err, msg_err) permet de déclencher une erreur utilisateur (avec num_err compris entre -20001 et -20999) et empêcher l'instruction initiale (qui a déclenché le trigger) d'aboutir.

Exemple : RAISE_APPLICATION_ERROR (-20222,'Mon message erreur');

3. Tester les insertions suivantes :
   - Réservation pour le client 100 de la chambre 15 pour 15 personnes : Elle doit être refusée.
     `INSERT INTO planning VALUES (15, NULL, 100, 15);`
   - Réservation pour le client 100 de la chambre 15 pour 2 personnes : Elle doit être insérée.
     `INSERT INTO planning VALUES (15, NULL, 100, 2);`

<u>Exercice 4</u> : Trigger 4 (Déclencheur Ligne LMD ligne).

a) Mettre à jour le montant de la remise la table LIGNE_FACTURE (**UPDATE**) qui correspond au résultat de l'opération suivante : mnt * qte * remise_pourcent/100 (ou 0 si pas remise)

b) Modifier la table **FACTURE** pour ajouter un champ **Total** de type **NUMBER**. (**ALTER TABLE**)

c) Initialiser (**UPDATE**) pour chaque facture, ce champ **Total** avec le montant correspondant à la somme des différents montants (mnt) des lignes de factures associées, en prenant en compte la quantité (qte), la montant de la remise et le taux de tva. Arrondir à deux décimales. Afficher les montants.

CLIENT	FAC_ID	FAC_DATE	TOTAL
Daniel CHABAUD	359	31/01/07	61.59
Jean-Paul SPITHAKIS	1022	31/01/07	66.68
Carmen MARTINET	1071	31/01/07	61.59
…			
César DIEUDONNÉ	2500	30/10/07	78.94

d) Créer le déclencheur **TR_LG_Facture,** associé à la table **LIGNE_FACTURE**, qui met à jour :
   - le champ **remise_mnt** (dans le cas d'une insertion ou d'une modification)
   - le champ **Total** à chaque insertion, modification et suppression d'une ligne dans la table.

Au sein d'un déclencheur, des évènements de différentes natures peuvent être regroupés. Dans le corps du déclencheur il est possible de retrouver la nature de ce événement déclencheur :

`IF INSERTING THEN … [END IF]` s'exécute uniquement dans le cas d'une insertion
`IF UPDATING THEN … [END IF]` dans le cas d'une modification
`IF DELETING THEN … [END IF]` dans le cas d'une suppression

e) Tester :

- Ajout les lignes de factures suivantes (vérifier la maj du champ **TOTAL**).
  (Remarque : en 2007, le taux de TVA était de 19.6)

LIF_ID	FAC_ID	QTE	REMISE_POURCENT	MNT	TAUX_TVA
20100	2500	1	20	10	19.6
20101	2500	2	NULL	20	19.6

Le total de la facture 2500 passe de 78.94 à 136.35€.

- Modification de la quantité à 2 pour la ligne 1317 → le total de la facture 359 passe à 112.42€
- Suppression de la ligne 1317 → le montant de la facture 359 passe à 10.76€

Exercice 5 : Trigger 5 : tables mutantes, déclencheurs composés (facultatif)

1. Afficher le salaire moyen d'un agent d'entretien arrondi à la centaine.
2. Faire un déclencheur **TR_AGENT_V1** sur modification (INSERT, UPDATE) de la table AGENT_ENTRETIEN. Il s'agit d'interdire d'affecter à un agent d'entretien un salaire inférieur à la moyenne des salaires.
3. Tester en tentant d'affecter un salaire de 1000€ aux agents en poste. Que se passe-t-il ? Désactiver le déclencheur qui ne peut donc être utilisé.

Une table **en mutation** est une table sur laquelle un changement est en train de se faire. Cette erreur est rencontrée seulement dans les déclencheurs Lignes.
Il n'est pas possible d'interroger ou de modifier la table concernée par le déclencheur (appelée table mutante). A cause du changement en cours, son état devient (instable - inconsistant) et les pseudo-colonnes new et old ne sont pas accessibles.

4. Pour résoudre ce problème de tables mutantes, il est nécessaire de créer un **déclencheur composé**.
   Créer un déclencheur composé, **TR_AGENT_V2** qui vérifie, lors de l'ajout ou modification d'un agent d'entretien que son salaire est bien supérieur ou égal à la moyenne des salaires des agents.

Dans un déclencheur composé :
- La clause **FOR** remplace la clause BEFORE/AFTER d'un trigger normal et permet de définir l'évènement déclencheur
- **FOLLOWS** : le trigger sera déclenché après le ou les triggers cité(s)
- **Déclarations communes** : permet de définir variables, sous-programmes communs aux différentes sections.
- **Déclarations locales** : permet de définir variables, programmes etc. locaux à la section
- Le code comporte au maximum 4 **sections** : BEFORE STATEMENT / BEFORE EACH ROW / AFTER EACH ROW / AFTER STATEMENT

Ce déclencheur devra :

- Afficher le nombre d'agents d'entretien et calculer la moyenne des salaires avant insertion/modification (BEFORE STATEMENT),
- Afficher l'ancien salaire de l'agent modifié, et déclencher une erreur si le nouveau salaire ne répond pas au critère demandé (BEFORE EACH ROW),
- Afficher le nombre de lignes modifiées (AFTER STATEMENT).

5. Tester en augmentant de 10 % puis de 20% les agents actuellement en poste (sans date de départ)