SGBD TP7 :

PL/SQL : TRIGGERS

- Durée 1h30.
- Pour ce TP vous avez besoin de la BD GestCons (BD_GestCons.txt) et du fichier package_bloc.txt.
- Le compte redu de ce TP doit être sous la forme suivante : n°question, bloc PL/SQL, capture écran du résultat.

Travail à faire :

1. Soit le bloc PL/SQL ci dessous :

```
DECLARE
Consm boolean :=FALSE ;
Vpat patient.num_ss%TYPE:=12345 ;
Vpatient patient.num_ss%TYPE ;
------------données concernant la consultation à ajouter
vdatec consultation.date_cons%TYPE:='04/12/2018';
vheurec consultation.heure_cons%TYPE:=14.8;
vmed medecin.num_matricule%TYPE:=1;
-------------------
errConsm EXCEPTION;

BEGIN

SELECT num_ss INTO vpatient
FROM patient where num_ss= vpat ;

-- verifier si le medecin a déjà une consultation le même jour et à la même heure avec un
autre patient ou s'il n'est pas disponible.

Consm := Pa_GestCons.f_verif_consMed(vmed,vdatec, vheurec);
IF consm =TRUE THEN
 Pa_GestCons.p_ListConsMed (vmed,vdatec);

 RAISE errConsm;
ELSE
 INSERT INTO CONSULTATION VALUES (seqjoker.NEXTVAL, vmed,vpat, vdatec,vheurec, 35,NULL);

END IF ;
COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('Ce patient n''existe pas!!!');
  WHEN errConsm THEN DBMS_OUTPUT.PUT_LINE('Le Medecin n''est pas disponible à cette
  heure!!!');
END ;
```

1.1. Utiliser la fonction ***f_verif_patient*** dans le bloc PL/SQL ci dessus pour vérifier si le patient existe dans notre base ou pas.

```
DECLARE
Consm boolean :=FALSE ;
Vpat patient.num_ss%TYPE:=12345 ;

------------données concernant la consultation à ajouter
vdatec consultation.date_cons%TYPE:='12/04/2018';
vheurec consultation.heure_cons%TYPE:=14.8;
vmed medecin.num_matricule%TYPE:=1;
-------------------
Existpat BOOLEAN;
errConsm EXCEPTION;
errpat EXCEPTION;

BEGIN
```

```
   Existpat:=pa_GestCons.f_verif_patient(vpat);

   IF Existpat= FALSE THEN RAISE errpat;
   END IF;
   -- verifier si le medecin a déjà une consultation le même jour et à la même heure avec un
   autre patient ou s'il n'est pas disponible.

   Consm := Pa_GestCons.f_verif_consMed(vmed,vdatec, vheurec) ;

   IF consm =TRUE THEN
   Pa_GestCons.p_ListConsMed (vmed, vdatec);
   RAISE errConsm;

   ELSE
   INSERT INTO CONSULTATION VALUES (seqjoker.NEXTVAL, vmed,vpat, vdatec,vheurec, 35,NULL);

   END IF ;
   COMMIT;
   EXCEPTION
   WHEN errpat THEN DBMS_OUTPUT.PUT_LINE('Ce patient n''existe pas!!!');
   WHEN errConsm THEN DBMS_OUTPUT.PUT_LINE('Le Medecin n''estpas disponible à cette
   heure!!!');
END ;
```

2. Les contraintes sur la table ***consultation*** :

2.1. Suite à la suppression d'une consultation, il faut ajouter dans la table **`Consultation_annulee`** les informations suivantes : l'identifiant du médecin, l'identifiant du patient, la date et l'heure de la consultation. Créer un trigger **`trig_annulCons`** pour répondre à cette contrainte.

```
CREATE OR REPLACE TRIGGER trig_annulCons
AFTER DELETE
ON consultation
FOR EACH ROW

BEGIN
INSERT INTO consultation_annulee VALUES(:OLD.num_matricule, :OLD.num_ss, :OLD.date_cons,
:OLD.heure_cons) ;
END ;
```

```
--Tests--
sql> select * from Consultation_annulee;
sql> DELETE FROM consultation Where num_cons=16;
sql> select * from Consultation_annulee;
```

2.2. Aucune consultation ne peut être modifier si la date de consultation est antérieur à la date de la modification (la date system). Créer un trigger ***`trig_ModifCons`*** pour vérifier cette contrainte.

```
CREATE OR REPLACE TRIGGER trig_ModifCons
BEFORE UPDATE
ON consultation
FOR EACH ROW
--WHEN (OLD.date_cons <SYSDATE)
BEGIN
IF :OLD.date_cons < SYSDATE THEN
RAISE_APPLICATION_ERROR(-20102, 'La modification est impossible!!!') ;
END IF;
END ;
```

```
--Test—
sql>SELECT * FROM consultation ;
```

```
sql> UPDATE consultation SET num_matricule=2 WHERE num_cons=12;
sql>SELECT * FROM consultation ;
```

3. Les contraintes sur la table ***médicament*** et la table ***Lot_Medicament*** :

3.1. Les médicaments de la table lot_medicament qui sont supprimes doivent passer dans la table ***Lot_medicament_Perimer.*** Créer un trigger **`trig_suplotMedic`** pour répondre à cette contrainte.

```
CREATE OR REPLACE TRIGGER trig_suplotMedic
AFTER DELETE
ON Lot_medicament
FOR EACH ROW
BEGIN

INSERT INTO Lot_medicament_Perimer
VALUES(:OLD.ref_medic,:OLD.datef,:OLD.qte_lot,USER,CURRENT_TIMESTAMP);

END ;
-- avec l'interface SQL Commands on va pas avoir le nom du USER(ANONYMOUS)
-- avec SQL*PLUS on va avoir le nom du USER (ici TPSGBD)
```

ⓘ Date/heure

- Le type **DATE** permet de stocker des moments ponctuels, la précision est composée du siècle, de l'année, du mois, du jour, de l'heure, des minutes et des secondes.
- Le type **TIMESTAMP** est plus précis dans la définition d'un moment (fraction de seconde).
- Le type **TIMESTAMP WITH TIME ZONE** prend en compte les fuseaux horaires.
- Le type **TIMESTAMP WITH LOCAL TIME ZONE** permet de faire la dichotomie entre une heure côté serveur et une heure côté client.
- Le type **INTERVAL YEAR TO MONTH** permet d'extraire une différence entre deux moments avec une précision mois/année.

Les variables suivantes permettent de retrouver le moment de la session et le fuseau du serveur (si tant est qu'il soit déporté par rapport au client).

- **CURRENT_DATE :** date et heure de la session (format DATE) ;
- **LOCALTIMESTAMP :** date et heure de la session (format TIMESTAMP) ;
- **CURRENT_TIMESTAMP :** date et heure de la session de type TIMESTAMP WITH LOCAL TIME **ZONE.**
- **SYSTIMESTAMP :** date et heure du serveur (format TIMESTAMP WITH TIME ZONE) ;
- **DBTIMEZONE :** fuseau horaire du serveur (format VARCHAR2) ;
- **SESSIONTIMEZONE :** fuseau horaire de la session client (format VARCHAR2).

```
--Test--
sql> DELETE FROM lot_medicament WHERE ref_medic='12AS45';
sql>SELECT * FROM Lot_medicament_Perimer ;
sql>SELECT * FROM Lot_medicament ;
```

3.2. Lors de la mise à jour (seulement ajout et modification) de la table **`Lot_Medicament`**, il faut faire la mise à jour de la quantité des médicaments concernés dans la table médicament. Créer un trigger **`trig_MAJStockMedic`** pour verifier cette contrainte.

```
CREATE OR REPLACE TRIGGER trig_MAJStockMedic
AFTER INSERT OR DELETE OR UPDATE OF qte_lot
ON Lot_Medicament
FOR EACH ROW

BEGIN
```

```
IF INSERTING THEN
      UPDATE Medicament SET qtestock= qtestock + :NEW.qte_Lot
      WHERE ref_medic=:NEW.ref_medic;
END IF;
IF UPDATING THEN

            UPDATE Medicament SET qtestock= qtestock+(:OLD.qte_lot-:NEW.qte_lot)
            WHERE ref_medic=:OLD.ref_medic;
END IF;

IF DELETING THEN
UPDATE Medicament SET qtestock= qtestock -:OLD.qte_Lot
      WHERE ref_medic=:OLD.ref_medic;
ELSE
RAISE_APPLICATION_ERROR(-2014,'Ajout Impossible!');
END IF;

END;
```

```
--Test--
sql>INSERT INTO Lot_Medicament(ref_medic, dateF, qte_lot, unite,
remarque)VALUES('12AS45','06/20/2020',100,'boite',NULL);
sql>SELECT * FROM lot_medicament;
sql>SELECT * FROM medicament;
```

4. Soit la vue **`VMedic_disponibles`** :

```
CREATE VIEW VMedic_disponibles(reference,libelle,vignette, date_peremption, Quantite,unite,
Remarque) AS
SELECT M.ref_medic, libelle, vignette, dateF,Qte_lot,unite,LM.remarque
FROM Medicament M, Lot_medicament LM
WHERE M.ref_medic=LM.ref_medic
AND qte_lot>0;
```

4.1. Cette vue doit permettre à l'utilisateur d'ajouter un nouveau lot de médicament à un médicament qui existe déjà dans la table **`Medicament`**. Créer un trigger **`trig_VAjoutLotMedic`** pour répondre à cette contrainte.

ⓘ Avant l'ajout il faut vérifier que:

- le médicament (ref_medic) existe déjà dans la table médicament

- le lot de médicament (ref_med, datef) n'existe pas déjà dans la table lot_medicament

```
CREATE OR REPLACE TRIGGER trig_VAjoutLotMedic
INSTEAD OF INSERT ON VMedic_disponibles
FOR EACH ROW
DECLARE
vref NUMERIC;

BEGIN
-- vérifier si le medicament existe deja

SELECT COUNT(*) INTO vref FROM medicament WHERE ref_medic=:NEW.reference;
IF vref = 1 THEN
-- vérifier si le lot à ajouter esxiste deja
SELECT COUNT(*) INTO vref FROM lot_medicament WHERE ref_medic=:NEW.reference AND datef
=:NEW.date_peremption;

IF vref=0 THEN


INSERT INTO Lot_medicament (ref_medic,datef,qte_lot,unite,remarque) VALUES (:NEW.reference,
:NEW.date_peremption, :NEW.quantite,:NEW.unite, :NEW.remarque);
ELSE
```

```
  RAISE_APPLICATION_ERROR(-20015,'Ce lot de medicament existe deja!');
END IF;
ELSE
RAISE_APPLICATION_ERROR(-20014,'Ajout Impossible!');
END IF;


END;
```

```
--Test--

sql>INSERT INTO vmedic_disponibles VALUES
('12AS45','test','yes','06/06/2021',30,'boite','test');
-- verfifier que l'insertion à eu bien lieu dans la table lot_medicament
sql> SELECT * FROM lot_medicament ;
--verifier la mise à jour de la quantité en stock
sql>SELECT * FROM medicament ;
```

5. A chaque connexion et déconnection d'un utilisateur a notre schema, on voudrait insérer dans la table **`HistoriqueBD`** les données suivantes l'utilisateur, la date, l'heure, action ('logon', 'logoff'):

5.1. Créer un trigger **`trig_espionLogOn`** pour les utilisateurs qui se connectent.

```
CREATE OR REPLACE TRIGGER trig_espionLogOn
AFTER LOGON
ON SCHEMA
BEGIN
INSERT INTO HistoriqueBD VALUES( USER, current_TIMESTAMP, 'logon');
END;
```

5.2. Créer un trigger **`trig_espionLogOff`** pour les utilisateurs qui se déconnectent.

```
CREATE OR REPLACE TRIGGER trig_espionLogOn
BEFORE LOGOFF
ON SCHEMA
BEGIN
INSERT INTO HistoriqueBD VALUES( USER, current_TIMESTAMP, 'logoff');
END;
```