R3.07 - SQL dans un langage de programmation
Cours 2 - Compléments de PL/SQL

A. Ridard

Vannes
iut :
Université Bretagne Sud

A propos de ce document

- Pour naviguer dans le document, vous pouvez utiliser :
  - le menu (en haut à gauche)
  - l'icône en dessous du logo IUT
  - les différents liens
- Pour signaler une erreur, vous pouvez envoyer un message à l'adresse suivante : anthony.ridard@univ-ubs.fr

Plan du cours

Dans le TP1, pour gérer le fait qu'un client ne peut pas être conseillé par un agent portant le même nom que lui, nous avons programmé le déclencheur suivant :

```
CREATE OR REPLACE TRIGGER trig_nomsIdentiques
BEFORE INSERT OR UPDATE ON Client
FOR EACH ROW
DECLARE
     v_nomAgent Agent.nomAgent%TYPE;
BEGIN
     SELECT nomAgent INTO v_nomAgent
     FROM Agent
     WHERE numAgent = :NEW.sonAgent;

     IF (:NEW.nomClient = v_nomAgent) THEN
          RAISE_APPLICATION_ERROR(-20006,'Pb de nom');
     END IF;
END;
/
```

L'appel à la procédure **RAISE_APPLICATION_ERROR** interrompt le programme puis retourne le numéro (entier compris entre -20 000 et -20 999) et le message d'erreur qui peuvent être récupérés par l'environnement englobant (variables **SQLCODE** et **SQLERRM**).

Ainsi, le script de test :

```
INSERT INTO Agence VALUES(1,'01 00 00 00 01','adresse1');
INSERT INTO Agent VALUES(1,'nom1','prenomAgent1',2000,1,1);

INSERT INTO Client VALUES (1,'nom1','prenomClient1','adresseCient1
    ',TO_DATE('01/01/2000','DD/MM/YYYY'),1);
```

fournit le rapport d'erreur :

RESULT

```
ORA-20006: Pb de nom
ORA-06512: at "RIDARD.TRIG_NOMSIDENTIQUES", line 9
ORA-04088: error during execution of trigger 'RIDARD.
    TRIG_NOMSIDENTIQUES'
```

Que va-t-il se passer avec cette instruction ?

```
INSERT INTO Client VALUES (1,'nom1','prenomClient1','adresseCient1
    ',TO_DATE('01/01/2000','DD/MM/YYYY'),NULL);
```

Que va-t-il se passer avec cette instruction ?

```
INSERT INTO Client VALUES (1,'nom1','prenomClient1','adresseCient1
    ',TO_DATE('01/01/2000','DD/MM/YYYY'),NULL);
```

On obtient le rapport d'erreur :


```
ORA-01403: no data found
ORA-06512: at "RIDARD.TRIG_NOMSIDENTIQUES", line 4
ORA-04088: error during execution of trigger 'RIDARD.
    TRIG_NOMSIDENTIQUES'
```

Deux approches peuvent alors être envisagées :

- Chercher à éviter la levée d'exception (pas toujours possible)
- Traiter la levée d'exception

Pour éviter cette levée d'exception, on peut ajouter la clause **WHEN**

```
CREATE OR REPLACE TRIGGER trig_nomsIdentiques
BEFORE INSERT OR UPDATE ON Client
FOR EACH ROW
WHEN (NEW.sonAgent IS NOT NULL)
DECLARE
     v_nomAgent Agent.nomAgent%TYPE;
BEGIN
     SELECT nomAgent INTO v_nomAgent
     FROM Agent
     WHERE numAgent = :NEW.sonAgent;

     IF (:NEW.nomClient = v_nomAgent) THEN
          RAISE_APPLICATION_ERROR(-20006,'Pb de nom');
     END IF;
END;
/
```

Rappels sur la clause WHEN

- Elle n'est disponible qu'en présence de **FOR EACH ROW** c'est à dire dans un trigger de ligne
- La condition est une expression SQL avec éventuellement **NEW** ou **OLD** (sans les « : »), mais ne pouvant inclure ni requêtes ni fonctions PL/SQL

En reprenant notre insertion :

```
INSERT INTO Client VALUES (1,'nom1','prenomClient1','adresseCient1
    ',TO_DATE('01/01/2000','DD/MM/YYYY'),NULL);
```

On obtient cette fois le rapport d'erreur :


```
ORA-01400: cannot insert NULL into ("RIDARD"."CLIENT"."SONAGENT")
```

Savez-vous pourquoi ?

En reprenant notre insertion :

```
INSERT INTO Client VALUES (1,'nom1','prenomClient1','adresseCient1
    ',TO_DATE('01/01/2000','DD/MM/YYYY'),NULL);
```

On obtient cette fois le rapport d'erreur :


```
ORA-01400: cannot insert NULL into ("RIDARD"."CLIENT"."SONAGENT")
```

Savez-vous pourquoi ?

Ordre de vérification des contraintes

- Trigger de ligne BEFORE
- Contraintes d'intégrité (clés, existence, unicité et domaine)
- Trigger d'état AFTER

Pour traiter cette levée d'exception, on peut ajouter la partie **EXCEPTION**

```
CREATE OR REPLACE TRIGGER trig_nomsIdentiques
BEFORE INSERT OR UPDATE ON Client
FOR EACH ROW
DECLARE
     v_nomAgent Agent.nomAgent%TYPE;
BEGIN
     SELECT nomAgent INTO v_nomAgent
     FROM Agent
     WHERE numAgent = :NEW.sonAgent;

     IF (:NEW.nomClient = v_nomAgent) THEN
          RAISE_APPLICATION_ERROR(-20006,'Pb de nom');
     END IF;

     EXCEPTION

          WHEN NO_DATA_FOUND THEN
               DBMS_OUTPUT.PUT_LINE('agent inconnu');
END;
/
```

Fonction DBMS_OUTPUT.PUT_LINE

- Elle permet d'afficher une sortie
- Elle nécessite l'activation du package **DBMS_OUTPUT**

On active l'affichage :

```
SET SERVEROUTPUT ON
```

En reprenant notre insertion :

```
INSERT INTO Client VALUES (1,'nom1','prenomClient1','adresseCient1
    ',TO_DATE('01/01/2000','DD/MM/YYYY'),NULL);
```

On obtient maintenant la sortie :


```
agent inconnu
...
ORA-01400: cannot insert NULL into ("RIDARD"."CLIENT"."SONAGENT")
```

Que va-t-il se passer avec cette instruction ?

```
INSERT INTO Client VALUES (1,NULL,'prenomClient1','adresseCient1',
    TO_DATE('01/01/2000','DD/MM/YYYY'),1);
```

Que va-t-il se passer avec cette instruction ?

```
INSERT INTO Client VALUES (1,NULL,'prenomClient1','adresseCient1',
    TO_DATE('01/01/2000','DD/MM/YYYY'),1);
```

Et avec celle-ci ?

```
UPDATE Agent SET nomAgent = NULL WHERE numAgent = 1 ;
INSERT INTO Client VALUES (2,'nom2','prenomClient2','adresseCient2
    ',TO_DATE('01/01/2000','DD/MM/YYYY'),1);
```

Que va-t-il se passer avec cette instruction ?

```
INSERT INTO Client VALUES (1,NULL,'prenomClient1','adresseCient1',
    TO_DATE('01/01/2000','DD/MM/YYYY'),1);
```

Et avec celle-ci ?

```
UPDATE Agent SET nomAgent = NULL WHERE numAgent = 1 ;
INSERT INTO Client VALUES (2,'nom2','prenomClient2','adresseCient2
    ',TO_DATE('01/01/2000','DD/MM/YYYY'),1);
```

Ces manipulations de données sont valides

La logique PL/SQL possède 3 états : TRUE, FALSE et UNKNOWN

Dans le TP0, pour gérer le fait qu'un emplacement ne peut être réservé par deux bateaux différents le même jour, nous avons programmé le déclencheur suivant :

```
CREATE OR REPLACE TRIGGER trig_resMemeEmpl
AFTER INSERT ON Reservation
DECLARE
    v_nbPb NUMBER;
BEGIN

    SELECT COUNT(*) INTO v_nbPb
    FROM Reservation R1, Reservation R2
    WHERE R1.lEmplacement = R2.lEmplacement
    AND R1.idReservation < R2.idReservation
    AND R1.dateFin > R2.dateDebut;

    IF (v_nbPb > 0) THEN
        RAISE_APPLICATION_ERROR(-20002,'Cette manipulation
     provoque '||v_nbPb||' conflits');
    END IF;

END;
/
```

Pour décider, nous devons requêter la table mutante, c'est pourquoi nous avons retiré la clause **FOR EACH ROW** autrement dit programmé un trigger d'état !

On prépare le test avec les insertions suivantes :

```
DELETE FROM Reservation;
DELETE FROM Bateau;
DELETE FROM Emplacement;
DELETE FROM Proprietaire;

INSERT INTO Proprietaire VALUES (1,'Ridard','Anthony','anthony.
    ridard@univ-ubs.fr');
INSERT INTO Proprietaire VALUES (2,'Pham','Minh-Tan','minh-tan.
    pham@univ-ubs.fr');
INSERT INTO Proprietaire VALUES (3,'Kerbellec','Goulven','goulven.
    kerbellec@univ-ubs.fr');

INSERT INTO Bateau VALUES (1,NULL,8,1,NULL);
INSERT INTO Bateau VALUES (2,NULL,9,3,NULL);
INSERT INTO Bateau VALUES (3,NULL,9,2,NULL);

INSERT INTO Emplacement VALUES (1,12,30);
INSERT INTO Emplacement VALUES (2,10,20);
INSERT INTO Emplacement VALUES (3,10,15);
INSERT INTO Emplacement VALUES (4,10,15);
```

Alors, l'instruction :

```
INSERT ALL
    INTO Reservation VALUES (1,SYSDATE+2,SYSDATE+9,1,2)
    INTO Reservation VALUES (2,SYSDATE+1,SYSDATE+8,2,2)
    INTO Reservation VALUES (3,SYSDATE+1,SYSDATE+8,2,3)
    INTO Reservation VALUES (4,SYSDATE+6,SYSDATE+13,3,3)
SELECT * FROM DUAL;
```

fournit le rapport d'erreur :


```
ORA-20002: Cette manipulation provoque 2 conflits
ORA-06512: at "RIDARD.TRIG_RESMEMEEMPL", line 12
ORA-04088: error during execution of trigger 'RIDARD.
    TRIG_RESMEMEEMPL'
```

Écrivons le trigger sans le **COUNT**(*) :

```
CREATE OR REPLACE TRIGGER trig_resMemeEmpl
AFTER INSERT ON Reservation
DECLARE
     v_idReservation Reservation.idReservation%TYPE;
BEGIN

     SELECT R2.idReservation INTO v_idReservation
     FROM Reservation R1, Reservation R2
     WHERE R1.lEmplacement = R2.lEmplacement
     AND R1.idReservation < R2.idReservation
     AND R1.dateFin > R2.dateDebut;

     IF (v_idReservation > 0) THEN
          RAISE_APPLICATION_ERROR(-20002,'Au moins un conflit');
     END IF;

     EXCEPTION

          WHEN NO_DATA_FOUND THEN
               DBMS_OUTPUT.PUT_LINE('Pas de conflit');

END;
/
```

Alors, l'instruction :

```
INSERT ALL
    INTO Reservation VALUES (1,SYSDATE+2,SYSDATE+9,1,2)
    INTO Reservation VALUES (3,SYSDATE+1,SYSDATE+8,2,3)
SELECT * FROM DUAL;
```

fournit la sortie :


```
Pas de conflit
Pas de conflit

2 lignes inséré.
```

Par contre, l'instruction :

```
INSERT ALL
    INTO Reservation VALUES (1,SYSDATE+2,SYSDATE+9,1,2)
    INTO Reservation VALUES (2,SYSDATE+1,SYSDATE+8,2,2)
    INTO Reservation VALUES (3,SYSDATE+1,SYSDATE+8,2,3)
    INTO Reservation VALUES (4,SYSDATE+6,SYSDATE+13,3,3)
SELECT * FROM DUAL;
```

fournit le rapport d'erreur :


```
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at "RIDARD.TRIG_RESMEMEEMPL", line 5
ORA-04088: error during execution of trigger 'RIDARD.
    TRIG_RESMEMEEMPL'
```

Traitons cette levée d'exception **TOO_MANY_ROWS** :

```
CREATE OR REPLACE TRIGGER trig_resMemeEmpl
AFTER INSERT ON Reservation
DECLARE
    v_idReservation Reservation.idReservation%TYPE;
BEGIN

    SELECT R2.idReservation INTO v_idReservation
    FROM Reservation R1, Reservation R2
    WHERE R1.lEmplacement = R2.lEmplacement
    AND R1.idReservation < R2.idReservation
    AND R1.dateFin > R2.dateDebut;

    IF (v_idReservation > 0) THEN
        RAISE_APPLICATION_ERROR(-20002,'Au moins un conflit');
    END IF;

    EXCEPTION

        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Pas de conflit');

        WHEN TOO_MANY_ROWS THEN
            RAISE_APPLICATION_ERROR(-20002,'Au moins deux conflits
    ');

END;
/
```

Maintenant, l'instruction :

```
INSERT ALL
    INTO Reservation VALUES (1,SYSDATE+2,SYSDATE+9,1,2)
    INTO Reservation VALUES (2,SYSDATE+1,SYSDATE+8,2,2)
    INTO Reservation VALUES (3,SYSDATE+1,SYSDATE+8,2,3)
    INTO Reservation VALUES (4,SYSDATE+6,SYSDATE+13,3,3)
SELECT * FROM DUAL;
```

fournit le rapport d'erreur :


```
ORA-20002: Au moins deux conflits
ORA-06512: at "RIDARD.TRIG_RESMEMEEMPL", line 21
ORA-04088: error during execution of trigger 'RIDARD.
    TRIG_RESMEMEEMPL'
```

Pour afficher les réservations en conflit, nous allons utiliser un curseur :

```
CREATE OR REPLACE TRIGGER trig_resMemeEmpl
AFTER INSERT ON Reservation
DECLARE
    CURSOR cur_resaAvecPb IS
        SELECT R1.idReservation resa_1, R2.idReservation resa_2
        FROM Reservation R1, Reservation R2
        WHERE R1.lEmplacement = R2.lEmplacement
        AND R1.idReservation < R2.idReservation
        AND R1.dateFin > R2.dateDebut;

    v_resaAvecPb cur_resaAvecPb%ROWTYPE;
    v_nbPb NUMBER :=0;
BEGIN
    OPEN cur_resaAvecPb;
        LOOP
            FETCH cur_resaAvecPb INTO v_resaAvecPb;
            EXIT WHEN cur_resaAvecPb%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE('Conflit entre '||v_resaAvecPb.
     resa_1||' et '||v_resaAvecPb.resa_2||' !');
            v_nbPb := v_nbPb + 1;
        END LOOP;
    CLOSE cur_resaAvecPb;

    IF (v_nbPb > 0) THEN
        RAISE_APPLICATION_ERROR(-20002,'Au moins un conflit');
    END IF;
END;
/
```

Alors, l'instruction :

```
INSERT ALL
    INTO Reservation VALUES (1,SYSDATE+2,SYSDATE+9,1,2)
    INTO Reservation VALUES (2,SYSDATE+1,SYSDATE+8,2,2)
    INTO Reservation VALUES (3,SYSDATE+1,SYSDATE+8,2,3)
    INTO Reservation VALUES (4,SYSDATE+6,SYSDATE+13,3,3)
SELECT * FROM DUAL;
```

fournit la sortie :


```
Conflit entre 1 et 2 !
Conflit entre 3 et 4 !
...
ORA-20002: Au moins un conflit
ORA-06512: at "RIDARD.TRIG_RESMEMEEMPL", line 23
ORA-04088: error during execution of trigger 'RIDARD.
    TRIG_RESMEMEEMPL'
```

Lorsque l'on doit parcourir toutes les lignes du curseur (c'est le cas ici), il est beaucoup plus intéressant (pourquoi ?) d'utiliser une boucle **FOR** :

```
CREATE OR REPLACE TRIGGER trig_resMemeEmpl
AFTER INSERT ON Reservation
DECLARE
    CURSOR cur_resaAvecPb IS
        SELECT R1.idReservation resa_1, R2.idReservation resa_2
        FROM Reservation R1, Reservation R2
        WHERE R1.lEmplacement = R2.lEmplacement
        AND R1.idReservation < R2.idReservation
        AND R1.dateFin > R2.dateDebut;

    v_nbPb NUMBER :=0;
BEGIN

    FOR v_resaAvecPb IN cur_resaAvecPb
        LOOP
            DBMS_OUTPUT.PUT_LINE('Conflit entre '||v_resaAvecPb.
     resa_1||' et '||v_resaAvecPb.resa_2||' !');
            v_nbPb := v_nbPb + 1;
        END LOOP;

    IF (v_nbPb > 0) THEN
        RAISE_APPLICATION_ERROR(-20002,'Au moins un conflit');
    END IF;

END;
/
```

Là encore, l'instruction :

```
INSERT ALL
    INTO Reservation VALUES (1,SYSDATE+2,SYSDATE+9,1,2)
    INTO Reservation VALUES (2,SYSDATE+1,SYSDATE+8,2,2)
    INTO Reservation VALUES (3,SYSDATE+1,SYSDATE+8,2,3)
    INTO Reservation VALUES (4,SYSDATE+6,SYSDATE+13,3,3)
SELECT * FROM DUAL;
```

fournit la sortie :


```
Conflit entre 1 et 2 !
Conflit entre 3 et 4 !
...
ORA-20002: Au moins un conflit
ORA-06512: at "RIDARD.TRIG_RESMEMEEMPL", line 19
ORA-04088: error during execution of trigger 'RIDARD.
    TRIG_RESMEMEEMPL'
```

1. Pour faciliter l'accès au port à un maximum d'usagers, une réservation ne doit pas dépasser 7 jours, et un propriétaire ne peut pas effectuer deux réservations espacées de moins de 15 jours.

   Implémenter ces deux nouvelles contraintes sans retoucher au script de création de tables, et en prenant le soin de bien afficher les conflits pour la deuxième contrainte.

2. Écrire une procédure anonyme permettant d'afficher, pour chaque propriétaire trié dans l'ordre alphabétique du nom, le nombre de réservations effectuées (**éventuellement 0**) ?

Si on est amené à exécuter plusieurs fois une procédure, il vaut mieux la nommer !

Reprise de la question 2 de l'exercice précédent

```
CREATE OR REPLACE PROCEDURE proc_affichage IS

CURSOR cur_nbResa IS
    SELECT nomProprietaire, COUNT(idReservation) nbResa
    FROM Proprietaire
        LEFT JOIN Bateau ON idProprietaire = leProprietaire
            LEFT JOIN Reservation ON idBateau = leBateau
    GROUP BY idProprietaire, nomProprietaire

    ORDER BY nomProprietaire;

BEGIN
    FOR v_nbResa IN cur_nbResa
        LOOP
            DBMS_OUTPUT.PUT_LINE('Le propriétaire '||v_nbResa.
     nomProprietaire||' a effectué '||v_nbResa.nbResa||' ré
     servation(s).');
        END LOOP;
END;
/
```

Pour appeler une telle procédure, on peut le faire :

à l'aide de EXECUTE en dehors d'un bloc

```
EXECUTE proc_affichage ;
```

Ou bien :

directement dans un bloc (resp. une procédure, un trigger, ...)

```
BEGIN
    proc_affichage ;
END;
/
```

Dans les deux cas, on retrouve bien :


```
Le proprietaire Kerbellec a effectue 0 reservation(s).
Le proprietaire Pham a effectue 2 reservation(s).
Le proprietaire Ridard a effectue 3 reservation(s).
```

On peut évidemment paramétrer une procédure !

Procédure nommée avec paramètre(s)

```
CREATE OR REPLACE PROCEDURE proc_affichage_param ( n NUMBER ) IS

CURSOR cur_nbResa IS
    SELECT nomProprietaire, COUNT(idReservation) nbResa
    FROM Proprietaire
        LEFT JOIN Bateau ON idProprietaire = leProprietaire
            LEFT JOIN Reservation ON idBateau = leBateau
    GROUP BY idProprietaire, nomProprietaire
    HAVING COUNT(idReservation) >= n
    ORDER BY nomProprietaire;

BEGIN
    FOR v_nbResa IN cur_nbResa
        LOOP
            DBMS_OUTPUT.PUT_LINE('Le propriétaire '||v_nbResa.
     nomProprietaire||' a effectué '||v_nbResa.nbResa||' ré
     servation(s).');
        END LOOP;
END;
/
```

L'instruction :

```
EXECUTE proc_affichage_param(2) ;
```

fournit alors :


```
Le proprietaire Pham a effectue 2 reservation(s).
Le proprietaire Ridard a effectue 3 reservation(s).
```

Interactions avec la base

Au delà de l'extraction de données (**SELECT**), on peut aussi automatiser les autres manipulations de données (**INSERT**, **UPDATE** et **DELETE**).

Procédure permettant d'appliquer une modification tarifaire globale

```
CREATE OR REPLACE PROCEDURE proc_modif_cout ( n NUMBER ) IS

BEGIN

    UPDATE Emplacement
        SET coutJournalier = coutJournalier + n;

END;
/
```

Le script :

```
EXECUTE proc_modif_cout (5) ;

SELECT *
FROM Emplacement ;
```

fournit alors :

IDEMPLACEMENT	LONGUEUREMPLACEMENT	COUTJOURNALIER
1	12	35
2	10	25
3	10	20
4	10	20

On peut enfin déclarer des fonctions pour compléter toutes les fonctions déjà prédéfinies dans le SGBD.

Fonction qui retourne le nombre de jours depuis la dernière réservation du propriétaire fourni

```
CREATE OR REPLACE FUNCTION fct_since_last_resa
     (
     p_nomProprietaire Proprietaire.nomProprietaire%TYPE,
     p_prenomProprietaire Proprietaire.prenomProprietaire%TYPE
     )
     RETURN NUMBER
IS

v_result NUMBER;

BEGIN

     SELECT ROUND(SYSDATE - MAX(dateFin),0) INTO v_result
     FROM Proprietaire
          JOIN Bateau ON idProprietaire = leProprietaire
               JOIN Reservation ON idBateau = leBateau
     WHERE nomProprietaire = p_nomProprietaire
     AND prenomProprietaire = p_prenomProprietaire;

     RETURN v_result;

END;
/
```

L'instruction :

```
SELECT fct_since_last_resa('Ridard', 'Anthony')
FROM DUAL;
```

fournit alors :


```
FCT_SINCE_LAST_RESA('RIDARD','ANTHONY')
---------------------------------------
                                     25
```

Quant à l'instruction :

```
SELECT emailProprietaire, fct_since_last_resa(nomProprietaire,
    prenomProprietaire) since_last_resa
FROM Proprietaire;
```

elle fournit :

RESULT

EMAILPROPRIETAIRE	SINCE_LAST_RESA
anthony.ridard@univ-ubs.fr	25
minh-tan.pham@univ-ubs.fr	24
goulven.kerbellec@univ-ubs.fr	