M1 IDSM-Kharkiv – Année 2018-2019
Bases de données avancées
Exercices pratiques (4) : Procédures stockées, SQL dynamique
J. Darmont (http://eric.univ-lyon2.fr/~jdarmont/), 16/07/2018

Exercice 1 : Procédure stockée

1. Reprendre le bloc PL/SQL anonyme permettant d'afficher le nom et le prix des 5 produits les plus chers de la table DEMO_PRODUCT_INFO (exercice 3 de la 3e série) et le transformer en procédure stockée de nom TOPK.

2. Exécuter la procédure stockée.

3. Modifier la procédure stockée pour qu'elle affiche les *k* produits les plus chers.

4. Tester l'exécution de la procédure stockée avec plusieurs valeurs de *k* (dont des valeurs supérieures à 10).

Aide-mémoire : Définition de déclencheur

```
CREATE [OR REPLACE] TRIGGER nom_declencheur
     BEFORE | AFTER
     INSERT | DELETE | UPDATE     |     [INSERT] [[OR] DELETE] [[OR] UPDATE]
     ON nom_table
     [FOR EACH ROW]
     -- Bloc PL/SQL
```

Exercice 2 : Déclencheur de transformation automatique

1. Créer la table DEMO_STATES à partir du script SQL https://eric.univ-lyon2.fr/~jdarmont/docs/demo_states.sql.

Toutes les données doivent être en majuscules dans la table DEMO_STATES.

2. Définir un déclencheur avant insertion ou modification qui affiche pour l'instant seulement les valeurs de :NEW.ST et :NEW.STATE_NAME. Tester l'insertion et la modification de quelques n-uplets.

3. Ajouter au déclencheur la transformation en majuscules les valeurs de ST et STATE_NAME avec la fonction UPPER. Tester l'insertion et la modification de quelques n-uplets et vérifier le contenu de la table DEMO_STATE.

Exercice 3 : Requête dynamique imbriquée dans un curseur statique

Écrire une procédure stockée permettant de compter le nombre de n-uplets dans toutes les tables de votre catalogue système (vue système TAB (TNAME, TABTYPE…)). Exclure les vues (type VIEW) de ce calcul. Afficher le résultat trié par ordre alphabétique sous la forme « NOM_TABLE : NB_NUPLETS n-uplets ». Tester.

Exercice 4 : Curseur dynamique

Créer une procédure stockée nommée « deuxAtt » qui affiche les valeurs des deux premiers attributs (généralement la clé primaire et un attribut que l'on espère intéressant) de la table passée en paramètre de la procédure. Les attributs de vos tables sont répertoriés dans la vue système USER_TAB_COLUMNS (TABLE_NAME, COLUMN_NAME…).

Indications :

- Récupérer les *noms* des deux premiers attributs de la table à l'aide d'un curseur paramétré.
- Écrire la requête qui permet de récupérer les *valeurs* des deux premiers attributs sous la forme d'une chaîne de caractères stockée dans une variable. Afficher la requête pour vérification.
- Utiliser un curseur dynamique pour exécuter la requête, parcourir son résultat et afficher les valeurs des deux premiers attributs.

Exemple de résultat attendu pour la table EMP :

```
SELECT EMPNO, ENAME FROM EMP
7369, SMITH
7499, ALLEN
7521, WARD
7566, JONES
7654, MARTIN
7698, BLAKE
7782, CLARK
7839, KING
7844, TURNER
7900, JAMES
7902, FORD
7934, MILLER
```

Exercice 5 : Déclencheur pour contrainte d'intégrité dynamique

La table CLIBANQUE(idcli, nomcli, idconjoint#) a un attribut idconjoint est l'identifiant (idcli) du conjoint du client courant. Écrire un déclencheur avant insertion ou modification qui contrôle que le nom du/de la conjoint·e d'un·e client·e est le même que celui de ce tte client e. Si les deux noms sont différents, une erreur fatale doit interrompre l'insertion ou la modification des données (exception).

1. Définir la structure de la table CLIBANQUE en SQL ou à l'aide du menu *Design*.

2. Écrire le déclencheur et le créer.

3. Insérer et modifier quelques n-uplets dans la table CLIBANQUE. Conclusion ?

Correction

```
-- Exercice 1

CREATE OR REPLACE PROCEDURE topk(k NUMBER) IS
     CURSOR ranking IS SELECT product_name, list_price FROM demo_product_info
                       ORDER BY list_price DESC;
     p ranking%ROWTYPE;

BEGIN
     OPEN ranking;
     FETCH ranking INTO p; -- Premier produit
     WHILE ranking%FOUND AND ranking%ROWCOUNT <= k LOOP
          DBMS_OUTPUT.PUT_LINE(ranking%ROWCOUNT || ') ' || p.product_name ||
                               ' : ' || p.list_price || ' €');
          FETCH ranking INTO p; -- Produit suivant
     END LOOP;
     CLOSE ranking;
END;

-- Exécution sous SQL Developer
EXECUTE topk(6)

-- Exécution dans un bloc PL/SQL
BEGIN
     topk(20);
END;


-- Exercice 2

-- Déclencheur
CREATE OR REPLACE TRIGGER state_maj
     BEFORE INSERT OR UPDATE
     ON demo_states
     FOR EACH ROW

BEGIN
     DBMS_OUTPUT.PUT_LINE(:NEW.ST || ' : ' || :NEW.STATE_NAME);
     :NEW.ST := UPPER(:NEW.ST);
     :NEW.STATE_NAME := UPPER(:NEW.STATE_NAME);
END;

-- Tests
INSERT INTO demo_states VALUES('xX', 'Phantom State');
UPDATE demo_states SET state_name = '51st state' WHERE ST = 'XX';
SELECT * FROM demo_states;
```

-- Exercice 3

```
CREATE OR REPLACE PROCEDURE compte IS

     CURSOR tables IS SELECT tname FROM tab WHERE tabtype <> 'VIEW'
          ORDER BY tname;
     t tables%ROWTYPE;
     c INTEGER;

BEGIN
     FOR t IN tables LOOP
          EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM ' || t.tname INTO c;
          DBMS_OUTPUT.PUT(t.tname || ' : ' || c || ' n-uplets');
     END LOOP;
END;
```

-- Exercice 4

```
CREATE OR REPLACE PROCEDURE deuxAtt(nomtable VARCHAR) IS

     CURSOR cparam(nomt VARCHAR) IS SELECT column_name FROM user_tab_columns
                                    WHERE table_name = nomt;
     TYPE CursDyn IS REF CURSOR;
     cdyn CursDyn;
     att1 user_tab_columns.column_name%TYPE;
     att2 user_tab_columns.column_name%TYPE;
     rq VARCHAR(255);
     val1 VARCHAR(255);
     val2 VARCHAR(255);

BEGIN
     -- Récupération des attributs via le curseur paramétré
     OPEN cparam(nomtable);
     FETCH cparam INTO att1;
     FETCH cparam INTO att2;
     CLOSE cparam;
     -- Construction de la requête
     rq := 'SELECT ' || att1 || ', ' || att2 || ' FROM ' || nomtable;
     DBMS_OUTPUT.PUT_LINE(rq);
     -- Parcours du curseur dynamique
     OPEN cdyn FOR rq;
     FETCH cdyn into val1, val2;
     WHILE cdyn%FOUND LOOP
          DBMS_OUTPUT.PUT_LINE(val1 || ', ' || val2);
          FETCH cdyn into val1, val2;
     END LOOP;
     CLOSE cdyn;
END;
```

-- Exercice 5

```
CREATE OR REPLACE TRIGGER test_conjoints
     BEFORE INSERT OR UPDATE
     ON clibanque
     FOR EACH ROW

DECLARE
     nomconjoint clibanque.nomcli%TYPE;
     mauvaisconjoint EXCEPTION;

BEGIN
     IF :NEW.idconjoint IS NOT NULL THEN
          SELECT nomcli
               INTO nomconjoint
               FROM clibanque
               WHERE idcli = :NEW.idconjoint;
          IF nomconjoint <> :NEW.nomcli THEN
               RAISE mauvaisconjoint;
          END IF;
     END IF;

EXCEPTION
     WHEN mauvaisconjoint THEN RAISE_APPLICATION_ERROR (-20501, 'Mise à jour
                              impossible : les noms des conjoints diffèrent.');

END;
```