M1 Informatique – Année 2024-2025
Bases de données avancées
TD n° 2 (PL/pgSQL) : Curseurs explicites
J. Darmont (https://eric.univ-lyon2.fr/jdarmont/), 21/08/24

Exercice 1

1. Définir au préalable un type enregistrement tNuplet(texte, nombre) qui servira de type de n-uplet (nombre est un entier).

2. Télécharger et exécuter le script SQL indiqué à l'adresse suivante, qui crée la table DEMO_PRODUCT_INFO.

https://eric.univ-lyon2.fr/jdarmont/docs/pgDemoProductInfo.sql

3. Écrire une fonction PL/pgSQL nommée listeProd() qui retourne les noms (PRODUCT_NAME) et les prix (LIST_PRICE) des produits de la table DEMO_PRODUCT_INFO à l'aide d'un *curseur explicite* (c'est uniquement pour l'exercice : comme on parcourt toute la table, on utiliserait en temps normal un curseur implicite).

4. Tester !

Exercice 2

1. Copier/coller la fonction listeProd() et la renommer top5().

2. Il s'agit maintenant de retourner les noms (PRODUCT_NAME) et les prix (LIST_PRICE) des 5 produits les plus chers de la table DEMO_PRODUCT_INFO. Pour cela :

- dans le curseur explicite, lister les produits par ordre de prix décroissant (la clause LIMIT est interdite dans la requête pour cet exercice, mais on l'emploierait en temps normal) ;
- modifier la condition d'arrêt du curseur pour qu'il continue tant que 5 n-uplets n'ont pas été lus (bien conserver la condition FOUND au cas où la table contienne moins de 5 n-uplets). Une variable entière sera utile pour compter les n-uplets lus !

3.Tester !

Exercice 3

Afin d'établir une corrélation, on souhaite connaître la différence de quantité moyenne entre les commandes successivement enregistrées dans une table de commandes (ORD). La table ORD est remplie de commandes valorisées (c'est-à-dire, pour lesquelles l'attribut quantité QTY n'est pas NULL) ou non. Les commandes non valorisées ne sont pas à prendre en compte. Écrire une fonction PL/pgSQL qui retourne la différence de quantité moyenne entre les commandes. Pour simplifier, on réduit la table ORD à l'unique attribut QTY.

Exemple :

QTY
5
NULL
10
8
9
13

Résultat attendu = (|10 – 5| + |8 – 10| + |9 – 8| + |13 – 9|) / 4 = 3

Indications :

- Créer la table ORD à l'aide du script https://eric.univ-lyon2.fr/jdarmont/docs/pgOrdQty.sql.

- Créer un curseur contenant les quantités des commandes valorisées (…`WHERE qty IS NOT NULL`).
- À l'aide d'un parcours explicite du curseur, lire la première quantité puis, pour toutes les quantités suivantes, cumuler la valeur absolue (fonction ABS) de *quantité courante – quantité précédente*.
- Exception : Interrompre le programme immédiatement si la table ORD contient moins de deux commandes valorisées. Pour ce type de traitements, on utilise habituellement une requête `COUNT`.

Exercice 4

On souhaite échantillonner la table EMP du TD n° 1. Écrire une fonction PL/pgSQL qui parcourt la table EMP et retourne les noms (ENAME) des employés (triés par numéro d'employé EMPNO) dont les rangs d'apparition dans la table sont : 1er, 3e, 6e, 10e, 15e… Indiquer également le rang dans la valeur de retour de la fonction.

Règle de calcul des rangs :

```
 1 =  0 + 1
 3 =  1 + 2
 6 =  3 + 3
10 =  6 + 4
15 = 10 + 5
…
```

Résultat attendu :

```
1    SMITH
3    WARD
6    BLAKE
10   JAMES
```

Indications :

- Parcourir un curseur explicite dans lequel n – 1 n-uplets sont sautés (instruction MOVE FORWARD *pas* FROM *curseur*) avant de lire un employé (avec n = 1, 2, 3… le nombre indiqué en gras dans le calcul des rangs ; les rangs sont soulignés).
- Réutiliser le type enregistrement tNuplet de l'Exercice 1.

Correction

```
-- Exercice 1

CREATE TYPE tNuplet AS (
     texte VARCHAR,
     nombre INT
);

CREATE OR REPLACE FUNCTION listeProd() RETURNS SETOF tNuplet AS $$
     DECLARE
          cursProd CURSOR FOR
               SELECT product_name, list_price FROM demo_product_info;
          nuplet tNuplet;
     BEGIN
          OPEN cursProd;
          FETCH cursProd INTO nuplet;
          WHILE FOUND LOOP
               RETURN NEXT nuplet;
               FETCH cursProd INTO nuplet;
          END LOOP;
          CLOSE cursProd;
          RETURN;
     END
$$ LANGUAGE plpgsql;

SELECT * from listeProd();


-- Exercice 2

CREATE OR REPLACE FUNCTION top5() RETURNS SETOF tNuplet AS $$
     DECLARE
          cursProd CURSOR FOR
               SELECT product_name, list_price FROM demo_product_info
               ORDER BY list_price DESC;
          nuplet tNuplet;
          n INTEGER := 0;
     BEGIN
          OPEN cursProd;
          FETCH cursProd INTO nuplet;
          WHILE FOUND AND n < 5 LOOP
               RETURN NEXT nuplet;
               n := n + 1;
               FETCH cursProd INTO nuplet;
          END LOOP;
          CLOSE cursProd;
          RETURN;
     END
$$ LANGUAGE plpgsql;

SELECT * from top5();


-- Exercice 3

CREATE OR REPLACE FUNCTION moyCom() RETURNS REAL AS $$
     DECLARE
          comVal CURSOR FOR SELECT qty FROM ord WHERE qty IS NOT NULL;
          qtePrec INTEGER; -- Quantité précédente
          qteCour INTEGER; -- Quantité courante
          total INTEGER := 0;
          n INTEGER;
```

```
     BEGIN
          -- Test du nombre de commandes valorisées
          SELECT COUNT(*) INTO n FROM ord WHERE qty IS NOT NULL;
          IF n < 2 THEN
               RAISE EXCEPTION 'Pas assez de commandes !';
          END IF;

          OPEN comVal;
          -- 1re quantité précédente
          FETCH comVal INTO qtePrec;
          -- 1re quantité courante
          FETCH comVal INTO qteCour;
          -- On peut cumuler, maintenant !
          WHILE FOUND LOOP
               total := total + ABS(qteCour - QtePrec);
               qtePrec := qteCour; -- La quantité courante devient la précédente
               FETCH comVal INTO qteCour; -- Quantité courante suivante
          END LOOP;
          CLOSE comVal;
          RETURN total / (n - 1) ::REAL;
     END
$$ LANGUAGE plpgsql;

SELECT moyCom();
```

-- Exercice 4

```
CREATE OR REPLACE FUNCTION rangEmp() RETURNS SETOF tNuplet AS $$
     DECLARE
          cursEmp CURSOR FOR SELECT ename FROM emp ORDER BY empno;
          res tNuplet;
          nom VARCHAR;
          n INTEGER := 1;
          rang INTEGER := 1;
     BEGIN
          OPEN cursEmp;
          FETCH cursEmp INTO nom;
          WHILE FOUND LOOP
               -- Retour du résultat
               res.nombre := rang;
               res.texte := nom;
               RETURN NEXT res;
               -- Incrémentation de n
               n := n + 1;
               -- Déplacement dans le curseur
               MOVE FORWARD n - 1 FROM cursEmp;
               -- Mise à jour du rang
               rang := rang + n;
               -- Employé suivant
               FETCH cursEmp INTO nom;
          END LOOP;
          CLOSE cursEmp;
          RETURN;
     END
$$ LANGUAGE plpgsql;

SELECT * from rangEmp();
```