SGBD : PROGRAMMATION ET ADMINISTRATION DES BASES DE DONNÉES [M2106]

TD(TP) $N^0 1 - 2$ - PROGRAMMATION AVANCÉE EN SQL

Objectifs

- Programmation avancée en SQL
- Fonctions Stockées en SQL

Enoncés

Exercice I : case, array

Dans un langage procédure, une permutation de deux données `a` et `b` peut être exprimée par le code suivant :

```
1   temp=a;
2   a=b;
3   b=temp;
```

Question 1.1. *Ecrire une fonction SQL* `permute` *qui permet de permuter les valeurs de deux données* `a` *et* `b` *transmises en paramètres, comme dans l'exemple :*

```
(abir) [abir] => select a,b from permute(1,2);
 a | b
---+---
 2 | 1
(1 row)
```

On s'intéresse maintenant à la permutation de deux valeurs consécutives de rang `j-1` et `j` d'un tableau `t` ayant un nombre effectif de `n` éléments, comme décrit par le code procédural suivant :

```
1   if (j>=2 && j<=n) -- limites
2   {
3         if (t[j-1] > t[j]) -- ordre decroissant
4         {
5               temp=t[j];      -- permuter
```

Date: 25 janvier 2014.

Hocine ABIR - IUT Villetaneuse .

```
6               t[j]=t[j-1];
7               t[j-1]=temp;
8           }
9       }
```

Question 1.2. *Ecrire une fonction SQL* `arp` *qui prend en entrée deux paramètres :*

(1) *un tableau* `t` *d'entiers*

(2) *un entier* `j` *(indice dans* `t`*)*

et qui retourne le tableau `t` *après avoir permuter les entrées* `j-1` *et* `j` *de* `t` *comme indiqué par le code ci-dessus.*

Exercice II : tri à bulle

Dans cet exercice, nous allons nous intéresser au tri à bulle dont le principe est résumé dans le code suivant pour un tableau `t` ayant `n` éléments :

```
1  for (int i=n; i<=2;i--)
2  {
3      for (int j=2;j<=i;j++)
4      {
5           if (t[j-1] > t[j]) -- ordre decroissant
6           {
7                temp=t[j];      -- permuter
8                t[j]=t[j-1];
9                t[j-1]=temp;
10          }
11     }
12 }
```

On considère la fonction SQL `nestedloop` suivante :

```
1  CREATE FUNCTION nestedloop(int [],
2                             out i int,out j int)
3   RETURNS SETOF record AS
4  $$
5  WITH RECURSIVE inl(k,f) AS
6   (
7       SELECT array_upper($1,1),2
8       UNION
9       SELECT k ,f+1
10           FROM inl
11            WHERE f<k
12  ),
```

```
13    outl (i,j) AS
14   (
15      SELECT k,f FROM inl
16      union
17      SELECT i-1 , j
18           FROM outl
19            WHERE i>j
20  )
21  SELECT * FROM outl;
22 $$ language SQL;
```

Question 2.1.

2.1.1. *Créer la fonction* `nestedloop` *en exécutant la commande ci-dessus.*

2.1.2. *Exécuter la requête suivante :*

```
select * from nestedloop(array[5,2,3,6,7]);
```

et étudier les résultats obtenus.

Question 2.2. *Décrire une fonction* `tribulle` *qui permet de trier un tableau d'entier en utilisant les boucles imbriquées (voir la question 2.1) comme dans l'exemple suivant :*

```
(abir) [abir] => select tribulle(array[7,5,2,4,3,1,6]);
    tribulle
-----------------
 {1,2,3,4,5,6,7}
(1 row)
```

Suggestion : décrire d'abord une fonction `bulle` *qui sera appelée par* `tribulle`*. Pensez à utiliser la fonction* `arp` *de l'exercice précédent.*

Exercice III : cte

On considère la cte `pyramide` suivante :

```
1  /*
2    Pyramide
3  */
4   WITH RECURSIVE pyramide (h) AS
5   (
6       SELECT 1
7       UNION
8       SELECT h + 1
9            FROM pyramide
```

```
10          WHERE h < 4
11  )
12  SELECT repeat(' ', 4-h) || -- Blancs
13         repeat('*', 2*h-1) as "Pyramide" -- Etoiles
14   FROM pyramide;
```

Question 3.1. *Reécrire cette requête sans utiliser le deuxième appel*

`repeat('*', 2*h-1) as "Pyramide"  -- Etoiles`

de la fonction `repeat`.

Question 3.2. *Reécrire la requête obtenue à la Question 3.1 sans utiliser le premier appel*

`repeat(' ', 4-h)` */ – Blancs/*

de la fonction `repeat`.

Question 3.3. *Modifier Votre requête pour afficher la pyramide inversée comme dans l'exemple ci-dessous :*

```
 Pyramide
----------
 *******
  *****
   ***
    *
(4 rows
```

Question 3.4. *A partir de la requête obtenue à la question 3.2, décrire une fonction SQL qui permet d'obtenir une pyramide de hauteur arbitraire, comme dans l'exemple suivant :*

```
(abir) [abir] => select pyramid(3);
 pyramid
---------
   *
  ***
 *****
(3 rows)

(abir) [abir] => select pyramid(5);
  pyramid
-----------
     *
    ***
   *****
```

```
  *******
 *********
(5 rows)
```

Corrigés

Exercice I :

Question 1.1.

```
CREATE FUNCTION permute(inout int a,inout int b)
 as
 $$
    SELECT $2,$1;
 $$ language SQL;
```

Question 1.2.

```
CREATE FUNCTION arp(inout int [],j int)
AS
$$
   SELECT case when $2>array_upper($1,1) or $2<2 then NULL
       else
          case when $1[$2-1]<=$1[$2] then $1
            else
                case when $2=2
                then  -- rien au debut
                 array_cat(array_append(
                     array_fill ($1[$2],array[1]),$1[$2-1]),
                                $1[$2+1:array_upper($1,1)])
               else
                   case when $2=array_upper($1,1)
                   then  -- rien a la fin
                        array_cat(
                           $1[1:$2-2],
                          array_append(
                           array_fill ($1[$2],array[1]),$1[$2-1])
                                )
                   else   -- cas general
                     array_cat(
                        array_cat(
                           $1[1:$2-2],
                           array_append(
                            array_fill ($1[$2],array[1]),$1[$2-1])
                                )
                        ,$1[$2+1:array_upper($1,1)]
                               )
                   end
               end
            end
       end
```

```
34 $$ language SQL;
```

Exercice II :

Question 2.1.

Question 2.2.

```
(abir) [abir] => select * from nestedloop(array[5,2,3,6,7]);
 i | j
---+---
 5 | 2
 5 | 3
 5 | 4
 5 | 5
 4 | 2
 4 | 3
 4 | 4
 3 | 2
 3 | 3
 2 | 2
(10 rows)
```

affiche les éléments du tableau qu'il faut comparer durant le tri à bulle.

Question 2.3.

```
1  CREATE FUNCTION bulle(int [],d int)
2   returns int[]
3  AS
4  $$
5  WITH RECURSIVE inl(k,f,s) AS
6   (
7       SELECT $2,2,
8                           array_to_string(arp($1,2),',')
9
10      UNION
11      SELECT k ,f+1,array_to_string(
12             arp(cast(string_to_array(s,',') as int[]),
13                       f+1),',')
14           FROM inl
15            WHERE f<k
16   )
17   SELECT cast(string_to_array(s,',')as int[]) FROM inl
```

```
    WHERE k=f;
$$ language SQL;
```

```
CREATE FUNCTION tribulle(int[])
 returns int[]
AS
$$
WITH RECURSIVE outl(i,t) AS
  (
     SELECT array_upper($1,1),
        array_to_string ( bulle ($1,array_upper($1,1)),',')
     union
     SELECT i-1 ,
     array_to_string ( bulle (cast(string_to_array(t,',')
                      as int[]),i-1),',')
        FROM outl
           WHERE i>2
  )
SELECT cast(string_to_array(t,',')as int[]) FROM outl
    WHERE i=2;
$$ language SQL
```

Exercice III :

Question 3.1.

```
 WITH RECURSIVE pyramide (h,l) AS
 (
     SELECT 1,'*'
     UNION
     SELECT h + 1, l||'**'
          FROM pyramide
           WHERE h < 4
 )
 SELECT repeat(' ', 4-h) ||| "Pyramide" FROM pyramide;
```

```
 Pyramide
----------
    *
   ***
  *****
 *******
(4 rows)
```

Question 3.2.

```
1  WITH RECURSIVE
2   pyramide1 (i,l) AS
3   (
4       SELECT 1,'*'
5       UNION
6       SELECT i+1, l||'**'
7            FROM pyramide1
8             WHERE i < 4
9   ),
10 pyramide2 (i,l) AS
11  (
12      SELECT 4,''
13      UNION
14      SELECT i-1, l||' '
15           FROM pyramide2
16            WHERE i > 1
17  )
18  SELECT p2.l||p1.l "Pyramide"
19     FROM pyramide1 p1, pyramide2 p2
20     Where p1.i=p2.i ;
```

```
 Pyramide
----------
    *
   ***
  *****
 *******
(4 rows)
```

Question 3.3.

```
1  WITH RECURSIVE
2   pyramide1 (i,l) AS
3   (
4       SELECT 1,'*'
5       UNION
6       SELECT i+1, l||'**'
7            FROM pyramide1
8             WHERE i < 4
9   ),
10 pyramide2 (i,l) AS
11  (
12      SELECT 4,''
```

```
    UNION
    SELECT i-1, |||' '
         FROM pyramide2
          WHERE i > 1
)
SELECT p2.|||p1.| "Pyramide"
   FROM pyramide1 p1, pyramide2 p2
   Where p1.i=p2.i
   order by p1.i desc ;
```

```
 Pyramide
----------
 *******
  *****
   ***
    *
(4 rows
```

Question 3.4.

```
CREATE or replace FUNCTION pyramid(int)
RETURNS SETOF varchar AS
$$
WITH RECURSIVE
 pyramide1 (i,|) AS
 (
    SELECT 1,'*'
    UNION
    SELECT i+1, |||'**'
         FROM pyramide1
          WHERE i < $1
 ),
pyramide2 (i,|) AS
 (
    SELECT $1,''
    UNION
    SELECT i-1, |||' '
         FROM pyramide2
          WHERE i > 1
 )
 SELECT p2.|||p1.| "Pyramide"
   FROM pyramide1 p1, pyramide2 p2
   Where p1.i=p2.i ;
$$ language SQL;
```