Databases

PL/pgSQL – Cursors

Marco Vieira

Bachelor in Informatics Engineering
Department of Informatics Engineering
University of Coimbra
2020/2021

From Previous Lesson(s)…

- The PL/pgSQL Language
- PL/pgSQL execution environment: anonymous blocks, procedures, functions, triggers
- Blocks and object scope
- Variables and basic data types: integer, numeric, varchar, ...
- SQL embedded in PL/pgSQL
- Control Structures: if, loop, while, for

Outline

- Cursors
- Implicit Cursors
- Operations with Explicit Cursors:
  - Open
  - Fetch
  - Close
- Locks in Cursors: FOR UPDATE
- WHERE CURRENT OF
- FOR LOOPS in Cursors

3

Cursors

- We need a working area to temporarily store the output of SQL instructions, in order to be able to process that output
- PL/pgSQL construction that allows assigning a name to that working area and access the data stored
- Two *types* of cursors :
  - Implicit cursors
  - Explicit cursors

4

Implicit Cursors

- Implicitly declared for DML and SELECT commands
- Key operations to evaluate the result of an implicit cursor: number of rows processed, rows found, rows not found

- Number of rows processed:

```
get diagnostics var := ROW_COUNT;
```

- Check if the last command processed any row:

```
if found then
```

- Check if the command did not process any row:

```
if not found then
```


5

Implicit Cursors: Example

```
do $$
declare
   deleted_lines numeric;
begin
   delete from emp
   where sal < 3000;

   if not found then
      insert into mytab values(-1);
   else
      get diagnostics deleted_lines := ROW_COUNT;

      insert into mytab
      values(deleted_lines);
   end if;
end;
$$;
```


6

Explicit Cursors

- Declared explicitly in a PL/pgSQL block
- Can only be used to SELECT
- Managed using specific instructions
- Allow processing several lines at once

- Four basic operations:
  - Declaration
  - Open
  - Fetch
  - Close

7

Declare Cursor

- Assigns a name to the cursor
- Define the SELECT to be execute
- It is possible to use arguments

declare
cursor_name **cursor** [(arguments)] **for**
select ...

```
declare
   c1 cursor for
      select nome, nemp
      from emp
      where sal > 2500;
   ...
```


8

Open Cursor

- Executes the SELECT
- The rows returned by the SELECT are ready to be fetched

open cursor_name [(arguments)];

```
declare
   c1 cursor for
      select nome, nemp
      from emp
      where sal > 2500;
   ...
begin
   open c1;
   ...
```

António Dias Neto	3252
Maria de Lurdes	3636
João Pinto	3333

9

Fetch Data From Cursor

- Loads the values of the current row into some variables
- Moves the pointer to the next row

fetch cursor_name
into var1, var2, ...;

```
declare
   c1 cursor for
      select nome, nemp
      from emp
      where sal > 2500;
   v_nome emp.nome%type;
   v_nemp emp.nemp%type;
begin
   open c1;
   ...
   fetch c1
   into v_nome, v_nemp;
   ...
```

António Dias Neto	3252
Maria de Lurdes	3636
João Pinto	3333

10

Close Cursor

- Frees the cursor data
- Cursor can be reopened afterwards

```
close cursor_name;
```

```
declare
   c1 cursor for
      select nome, nemp
      from emp
      where sal > 2500;
   ...
begin
   ...
   fetch c1
   into v_nome, v_nemp;
   ...
   close c1;
   ...
```


Control Access to Cursors

```
...
begin
   ...
   loop
      fetch c1 into v_nemp;
      exit when not found;
      ...
   end loop;
...
```

Cursors and Records

- It is possible to declare variables of the type record and %rowtype

```
declare
   c1 cursor for
      select nome, nemp
      from emp
      where sal > 2500;
   v_cur record;
begin
   open c1;
   ...
   fetch c1
   into v_cur;
   ...
   insert into mytab
   values (v_cur.nemp);
   ...
```

```
declare
   c1 cursor for
      select *
      from emp
      where sal > 2500;
   v_cur emp%rowtype;
begin
   open c1;
   ...
   fetch c1
   into v_cur;
   ...
   insert into mytab
   values (v_cur.nemp);
   ...
```


FOR UPDATE

- To lock a set of rows, we can use FOR UPDATE OF
- Locking is done when the cursor is opened

```
declare
   c1 cursor for
      select nome, nemp
      from emp
      where sal > 2500
      for update;
   ...
begin
   open c1;
   ...
```

WHERE CURRENT OF

- We can use the reference of the current line in the cursor in SQL commands

```
declare
   c1 cursor for
      select nome, nemp
      from emp
      where sal > 2500
      for update;
   v_cur record;
begin
   ...
   fetch c1 into v_cur;
   ...
   update emp set sal=sal*1.1
   where current of c1;
   ...
```


Cursors With Arguments

- Arguments can be used to pass values to cursors
- This allows the same cursor definition to correspond to different sets of data

```
declare
   c1 cursor (p_sal emp.sal%type) for
      select * from emp
      where sal > p_sal;
begin
   open c1(2000);
   ...
   close c1;
   ...
   open c1(3000);
   ...
```

FOR Loops and Cursors

- FOR loops automatically do:
  - Open cursor
  - FETCH one row in each interaction
  - Exit when all rows are processed
  - Close cursor

```
for var in cursor_name[(arguments)]
loop
    -- actions
    -- var is a record variable
end loop;
```

- The control variable is automatically declared

17

FOR Loops and Cursors: Example

```
do $$
declare
   c1 cursor for
      select * from emp
      where funcao = 'Vendedor' or sal > 2000
      for update;
      new_sal numeric;
begin
   for r in c1
   loop
      new_sal := (r.sal - 200) + r.sal*1.05;
      update emp set sal = new_sal
      where current of c1;
   end loop;
   commit;
end;
$$;
```


18

Q&A

19

Databases

PL/pgSQL - Cursors

Marco Vieira

Bachelor in Informatics Engineering
Department of Informatics Engineering
University of Coimbra
2020/2021

20