09. Programming Databases

Cursors

- Cursors are means of iterating through result-set of a query (one by one).
- It works like a pointer to a query result-set.
- Primary reason for using cursors is to avoid memory overrun when the result contains a large number of rows.
- FOR loop that we have already seen iterating through a result-set, is basically using a cursor, implicitly; also referred as implicit cursor.
- Cursor definition is a comprehensive and provides more control over the way result-set is "navigated" and "accessed".
- Cursor can be
    - Scrollable or Non-scrollable,
    - Updatable or Non-updatable
- **Scrollable Cursor**: if you can go back and forth in a resultset through the cursor, then it is scrollable cursor
- **Update Cursor**: if you can update the row being referred by the cursor, then it is "updatable cursor"
- In some procedural languages, by default cursors are read-only, and non-scrollable
- PL/PgSQL makes cursor updatable if possible, i.e. for simple (non-join, non-grouping) cursor queries.
- Default scrollable behavior is also query dependent.

Operations on Cursors

- Cursor has two components:
    - Cursor variable, and
    - Query associated with the cursor variables
- Declare a cursor variable
- Associate (bind) a query with cursor
- Open a cursor
- Fetch a row from the cursor
- Close a cursor

Three ways of creating cursor variables

(1) Unbound

(2) Bound

(3) Parameteric

DECLARE

All access to cursors in PL/pgSQL goes through cursor variables, which are always of the special data type **refcursor**. One way to create a cursor variable is just to declare it as a variable of type **refcursor** (unbound cursor). Another way is to use the cursor declaration syntax, creates bound cursor

name [[NO] SCROLL] CURSOR [(arguments)] FOR/IS query;

SCROLL/NO SCROLL to specify if cursor is scrollable, that means, you can scroll back.

Cursor can have arguments, which are actually specified while opening it

Examples:

```
curs1 refcursor;          --unbound cursor

curs2 CURSOR FOR      -- bound cursor
        SELECT * FROM employee;

curs3 CURSOR(pdno integer) IS          -- parametric cursor
        SELECT * FROM employee WHERE dno = pdno;
```

- If you associate a query with cursors at declaration time itself, then it is bound cursor, otherwise it is unbound.
- You associate query with unbound at the time of opening it. You can associate another query with such cursor variables, once done with earlier query.

Opening Unbound Cursors – specify the query at opening time

```
OPEN unbound_cursor [ [ NO ] SCROLL ] FOR query;
```

An example: `OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;`

Where curs1 has been declared as following: `curs1 refcursor;`

Opening Unbound Cursors for dynamic queries

OPEN FOR EXECUTE: is the command for queries that are created at run-time.

General syntax: `OPEN unbound_cursor [ [ NO ] SCROLL ] FOR EXECUTE query_string;`

An example: OPEN curs1 FOR EXECUTE 'SELECT * FROM ' || table_name;

Opening a Bound Cursor

General Syntax: OPEN bound_cursor [(argument_values)];

Examples:

```
curs2 CURSOR FOR SELECT * FROM employee;
OPEN curs2;

curs3 CURSOR(pdno integer) IS SELECT * FROM employee WHERE dno = pdno;
OPEN curs3(5);      --open cursor for dno=5
```

Fetching a row from cursor

Fetching does collection of data into variables from cursor. General syntax is:

```
FETCH [direction {FROM|IN}] cursor INTO target;
```

Examples (should be self-explanatory)-

```
FETCH curs1 INTO rowvar;  -- data of current row are read into a record type variable
FETCH curs2 INTO foo, bar, baz;
        --assumingly current row has three attributes are collected into respective variables
FETCH LAST FROM curs3 INTO x, y;  -- fetches data from last tuple
FETCH RELATIVE -2 FROM curs4 INTO x;
        -- fetches data from row that is -2 away from current position
```

Navigating through cursor rows without reading

Has move command for his purpose. General syntax is

```
MOVE [ direction { FROM | IN } ] cursor;
```

- Only the difference with FETCH is, the move, just moves the cursor to new location, we do not collect the row data
- Example-

```
MOVE curs1;
MOVE LAST FROM curs3;
MOVE RELATIVE -2 FROM curs4;
```

Example ##:

```
--Call: select  curs_emp(4);
CREATE or replace FUNCTION curs_emp(dno integer) RETURNS integer AS $$
DECLARE
  c CURSOR (pdno integer) IS SELECT fname, salary FROM employee WHERE dno = pdno;
  fname text;
  salary real;
BEGIN
        raise notice 'new run for dno=%', mdno;
        open c(mdno);
        LOOP
                FETCH c INTO fname, salary;
                EXIT WHEN NOT FOUND;
                raise notice '% %', fname, salary;
        END LOOP;
        CLOSE c;
        RETURN 1;
END  $$ LANGUAGE plpgsql;
```

Below is procedure for computing rank for admissions

```
--select compute_rank();
create or replace function compute_rank() RETURNS integer AS $$
DECLARE
        m_rank integer := 1;
        scur CURSOR FOR SELECT * FROM applications ORDER BY marks DESC;
        srec applications%ROWTYPE;

BEGIN
        OPEN scur;
        LOOP
                FETCH scur INTO srec;
                EXIT WHEN NOT FOUND;
                UPDATE applications SET da_rank = m_rank WHERE appno = srec.appno;
                                --can use CURRENT OF scur if cursor is updatable;
                m_rank := m_rank + 1;
        END LOOP;
        CLOSE scur;
        return 1;
END $$ LANGUAGE plpgsql;
```