Relational Programming (PL/SQL)

Components of the language:

- Simple variables
- Relation variables
- Procedures and Functions
- Cursors
- Statements:
  - Simple variable declaration (DECLARE, VAR)
  - Relation variable declaration (CREATE)
  - Procedures and Functions declarations (CREATE OR REPLACE)
  - Cursor declaration
  - Garbage collection for relation variables (DROP)
  - Garbage collection for procedures and functions (DROP)
  - Assignment statements to simple variables
  - Assignment statements to relation variables (INSERT, DELETE, UPDATE)
  - Block statements
  - Loop statements
  - Conditional statements
  - Procedure and function calls
  - Cursor operations (OPEN, FETCH, CLOSE)

Look at

http://www.utexas.edu/its/unix/reference/oracledocs/v92/B10501_01/appdev.920/a96624/01_oview.htm

http://www.csee.umbc.edu/help/oracle8/server.815/a67842/toc.htm
}

```
/*   Compute and store the transitive closure of GRAPH in a temporary*/
/*   table, TC.  In the algorithm, we will need an additional */
/*   temporary table, TCNEW */

/* relation variables declaration */
create table TC    (V1 INT, V2 INT);
create table TCNEW (V1 INT, V2 INT);

/* procedure TClosure computes the transitive closure of GRAPH and */
/* stores the result in TC */
/* TClosure does not have parameters  */

/* procedure declaration */
create or replace procedure TClosure as
  /* local simple variables declaration */
  num_new_edges INT;
  /* body of the procedure */
  begin
   /*  relation variable assignment */
   insert into TCNEW select * from GRAPH;

   /*  simple variable assignment using an SQL statement  */
   select COUNT(*) into num_new_edges from TCNEW;
   /*  loop statement */
   while num_new_edges > 0
    loop
      insert into TC select * from TCNEW;
      delete from TCNEW;
      insert into TCNEW  select T.V1, G.V2
                         from TC T, GRAPH G
                         where T.V2 = G.V1;

       select COUNT(*) into num_new_edges from
                 (select * from TCNEW
                  EXCEPT
                  select * from TC);
     end loop;
  end;
```

```
/* Main program block statement */
begin
  TClosure();
end;

/* Take input. */
variable vertex NUM;
accept vert prompt "Enter vertex: ";
select TC.V2 from TC where TC.V1 = &vertex;

drop table TCNEW;
drop table TC;
drop procedure TClosure;
```

Declaring a Cursor

```
CURSOR cursor_name [(parameter[, parameter]...)]
   IS select_statement;

cursor_parameter_name datatype
```

```
DECLARE
   CURSOR c1 IS SELECT empno, ename, job, sal FROM emp
      WHERE sal > 2000;
```

A cursor can take parameters

```
DECLARE
   CURSOR c1 (low  INTEGER, high INTEGER) IS SELECT ...
```

Opening a Cursor

```
DECLARE
   CURSOR c1 IS SELECT ename, job FROM emp WHERE sal < 3000;
   ...
BEGIN
   OPEN c1;
   ...
END;
```

```
DECLARE
   emp_name emp.ename%TYPE;
   salary   emp.sal%TYPE;
   CURSOR c1 (name VARCHAR2, salary NUMBER) IS SELECT ...

OPEN c1(emp_name, 3000);
OPEN c1('ATTLEY', 1500);
OPEN c1(emp_name, salary);
```

Fetching with a Cursor

```
...
OPEN c1;
...
FETCH c1 INTO my_empno, my_ename, my_deptno;
```

```
Repeated fetching in a loop
...
OPEN c1;
...
LOOP
   FETCH c1 INTO my_empno, my_ename, my_deptno;
   EXIT WHEN c1%NOTFOUND;
   -- process data record
END LOOP;
```

Closing a cursor

```
DECLARE
   CURSOR c1 IS SELECT ename FROM emp;
   name emp.ename%TYPE;

BEGIN
   OPEN c1;
   FETCH c1 INTO name;
   ...
   CLOSE c1;
END;
```

Using Cursor FOR Loops

In most situations that require an explicit cursor, you can simplify coding by using a cursor FOR loop instead of the OPEN, FETCH, and CLOSE statements.

```
DECLARE
   result temp.col1%TYPE;
   CURSOR c1 IS
      SELECT n1, n2, n3 FROM data_table WHERE exper_num = 1;
BEGIN
   FOR c1_rec IN c1 LOOP
      /* calculate and store the results */
      result := c1_rec.n2 / (c1_rec.n1 + c1_rec.n3);
      INSERT INTO temp VALUES (result, NULL, NULL);
   END LOOP;
END;
```

Very simple example

```
DECLARE
    /* Output variables to hold the result of the query: */
    a T1.e%TYPE;
    b T1.f%TYPE;

    /* Cursor declaration: */
    CURSOR T1Cursor IS
       SELECT e, f
       FROM T1
       WHERE e < f

BEGIN
   OPEN T1Cursor;
   LOOP
       /* Retrieve each row of the result of the above query
              into PL/SQL variables: */
        FETCH T1Cursor INTO a, b;

        /* If there are no more rows to fetch, exit the loop: */
        EXIT WHEN T1Cursor%NOTFOUND;

        /* Insert the reverse tuple: */
        INSERT INTO T2 VALUES(b, a);
   END LOOP;

   /* Free cursor used by the query. */
   CLOSE T1Cursor;
END;
```