PL/SQL

Francisco Moreno
Universidad Nacional

Paquetes

- Un paquete es una agrupación de funciones, procedimientos y variables.
- Es posible tener incluso varios subprogramas con el mismo nombre siempre y cuando difieran en los tipos de los parámetros → Sobrecarga
- Un paquete tiene dos partes:
  - Especificación → Definiciones de variables “públicas” y prototipos de los subprogramas
  - Cuerpo (BODY) → Implementación de los subprogramas declarados en la especificación más subprogramas y variables privadas

Sintaxis

ESPECIFICACIÓN

```
CREATE PACKAGE nom_paquete IS
  -- Variables públicas
  -- Declaración de subprogramas
   (públicos)
END;
/
```

CUERPO

```
CREATE PACKAGE BODY nom_paquete IS
  -- Variables privadas
   /* Implementación de subprogramas
     privados */
   /* Implementación de subprogramas
     declarados en la especificación */
END;
/
```

Los subprogramas privados se deben implementar antes que los públicos

Ejemplo (con sobrecarga): Especificación

```
CREATE OR REPLACE PACKAGE mat IS
 TYPE t_num IS TABLE OF NUMBER(3)
  INDEX BY BINARY_INTEGER;
 FUNCTION suma(arr IN t_num) RETURN
  NUMBER;
 FUNCTION suma(a IN NUMBER, b IN NUMBER)
  RETURN NUMBER;
END; --Fin de la especificación
/
```

Ejemplo: Cuerpo

```
CREATE OR REPLACE PACKAGE BODY mat IS
FUNCTION suma(arr IN t_num) RETURN
NUMBER IS
i NUMBER := arr.FIRST;
suma NUMBER(8) := 0;
BEGIN
 WHILE i IS NOT NULL LOOP
  suma := suma + arr(i);
  i := arr.NEXT(i);
 END LOOP;
 RETURN suma;
END; --Fin de la función suma
FUNCTION suma(a IN NUMBER, b IN NUMBER) RETURN
NUMBER IS
BEGIN
 RETURN a+b;
END; --Fin de la función suma
END; --Fin del cuerpo del paquete
/
```

```
DECLARE
mi_arr mat.t_num;
BEGIN
mi_arr(10):= 14;
mi_arr(8):= 2;
mi_arr(-1):= 3;
DBMS_OUTPUT.PUT_LINE(mat.suma(mi_arr));
DBMS_OUTPUT.PUT_LINE(mat.suma(4,5));
END;
/
```

Variables en los Paquetes

- Las variables de los paquetes son persistentes

  durante la sesión
- Las variables declaradas en la **especificación** son públicas: se pueden acceder y modificar directamente desde otros subprogramas o bloques anónimos
- Las variables declaradas en el **cuerpo** son privadas:

  solo se pueden acceder dentro del cuerpo

  Si se desea ver su valor desde afuera del cuerpo, se debe crear un subprograma (público) a través del cual se acceden

Ejemplo

```
CREATE OR REPLACE PACKAGE vbles IS
    --Vble y función pública
    a NUMBER(3) := 0;
    FUNCTION get_b RETURN NUMBER;
END;
/
```

```
CREATE OR REPLACE PACKAGE BODY vbles IS
   b NUMBER(3) := 800; --Vble. privada
FUNCTION get_b RETURN NUMBER IS
   BEGIN
   RETURN b;
   END;
 END;
/
```

```
BEGIN
vbles.a := vbles.a + 1;
DBMS_OUTPUT.PUT_LINE(vbles.a);
DBMS_OUTPUT.PUT_LINE(vbles.get_b);
--DBMS_OUTPUT.PUT_LINE(vbles.b); --Error
END;
/
```

Cursores REF

- Los **cursores REF** permiten pasar los resultados de una consulta a otro subprograma
- El programa que recibe el **cursor** REF lo puede recorrer como un cursor normal
- Oracle define un tipo de datos para declarar estos cursores:

  SYS_REFCURSOR

Ejemplo cursor REF: **SYS_REFCURSOR**

```
CREATE OR REPLACE PROCEDURE devuelve_cursor
/*Se envía el cursor REF (parámetro de salida) y el
  nombre de la tabla a consultar */
  (c OUT SYS_REFCURSOR, nom_tabla IN VARCHAR)
AS
BEGIN
 OPEN c FOR 'SELECT * FROM ' || nom_tabla; --Se llena
END;
/
```

Nótese que la consulta asociada con un REF CURSOR puede ser ¡una cadena de caracteres!, esto permite crear cursores cuya consulta se desconoce en tiempo de compilación.

```
DROP TABLE emp;
CREATE TABLE emp(
 cedula NUMBER(8) PRIMARY KEY,
 nom VARCHAR2(10) NOT NULL
);
INSERT INTO emp VALUES(8,'Loui');
INSERT INTO emp VALUES(7,'Dina');
INSERT INTO emp VALUES(5,'Xena');

DROP TABLE cliente;
CREATE TABLE cliente(
 ced NUMBER(8) PRIMARY KEY,
 nom VARCHAR2(10) NOT NULL,
 ciudad VARCHAR2(10) NOT NULL
);
INSERT INTO cliente VALUES(1,'Pedra', 'Cali');
INSERT INTO cliente VALUES(2,'Kathy', 'Londres');
```

```
DECLARE
--Una variable de tipo SYS_REFCURSOR
mi_ref_cursor SYS_REFCURSOR;
a1 emp%ROWTYPE;
a2 cliente%ROWTYPE;
BEGIN
devuelve_cursor(mi_ref_cursor,'emp'); --Se invoca
LOOP
 FETCH mi_ref_cursor INTO a1;
 EXIT WHEN mi_ref_cursor%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE(a1.cedula || ' ' || a1.nom);
END LOOP;
CLOSE mi_ref_cursor;
```

El programa continua en la próxima diapositiva

```
devuelve_cursor(mi_ref_cursor, 'cliente'); --Se invoca
LOOP
 FETCH mi_ref_cursor INTO a2;
 EXIT WHEN mi_ref_cursor%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE(a2.ced || ' ' || a2.nom || ' '
   || a2.ciudad);
END LOOP;
CLOSE mi_ref_cursor;
END;
/
```

- ¿Se podrá adaptar el programa anterior de tal forma que quede funcionando para imprimir los resultados de cualquier consulta? El problema es que solo en tiempo de ejecución se conoce la tabla a consultar y sus columnas → Solución: Usar el paquete DBMS_SQL. Ver ejemplo en la página, menú Varios

La cláusula FORALL

- PL/SQL envía las sentencias DML al motor de ejecución de SQL.
- Se puede minimizar el costo de la comunicación entre PL/SQL and SQL mediante características de PL/SQL conocidas como *bulk SQL* (BULK COLLECT, FORALL) y su uso en conjunto con colecciones (arreglos) → *Turbocharged DML*.
- La cláusula FORALL envía sentencias INSERT, UPDATE, y DELETE en lotes, en vez de una a la vez, como lo hace un ciclo convencional.

Ejemplo

```
DROP TABLE p1;
DROP TABLE p2;
CREATE TABLE p1(pnom VARCHAR2(20));
CREATE TABLE p2(pnom VARCHAR2(20));

DECLARE
TYPE NomTab IS TABLE OF p1.pnom%TYPE INDEX BY
PLS_INTEGER;
pnoms NomTab;
iteraciones CONSTANT PLS_INTEGER := 25000;
t1 INTEGER;  t2 INTEGER;  t3 INTEGER;
```

Continúa

```
BEGIN
DELETE p1;
DELETE p2;
FOR j IN 1..iteraciones LOOP
    pnoms(j) := 'Parte No. ' || TO_CHAR(j);
END LOOP;
t1 := DBMS_UTILITY.GET_TIME;
FOR j IN 1..iteraciones LOOP  -- Se usa FOR
   INSERT INTO p1 VALUES (pnoms(j));
END LOOP;
```

Continúa

```
t2 := DBMS_UTILITY.GET_TIME;
FORALL j IN 1..iteraciones  -- Se usa FORALL
  INSERT INTO p2 VALUES (pnoms(j));
t3 := DBMS_UTILITY.GET_TIME;
DBMS_OUTPUT.PUT_LINE('Tiempo de ejecución (seg)');
DBMS_OUTPUT.PUT_LINE('FOR: ' || (t2 - t1)/100);
DBMS_OUTPUT.PUT_LINE('FORALL: ' || (t3 - t2)/100);
END;
/
```

Ejercicios: Ver que pasa si se coloca `'Parte No. ' || TO_CHAR(j)` en vez de `pnoms(j)` en el `INSERT` de `p2`

Ver otras opciones de `FORALL`: `INDICES OF` y `VALUES OF`