Facultad de Estudios Superiores
Acatlán

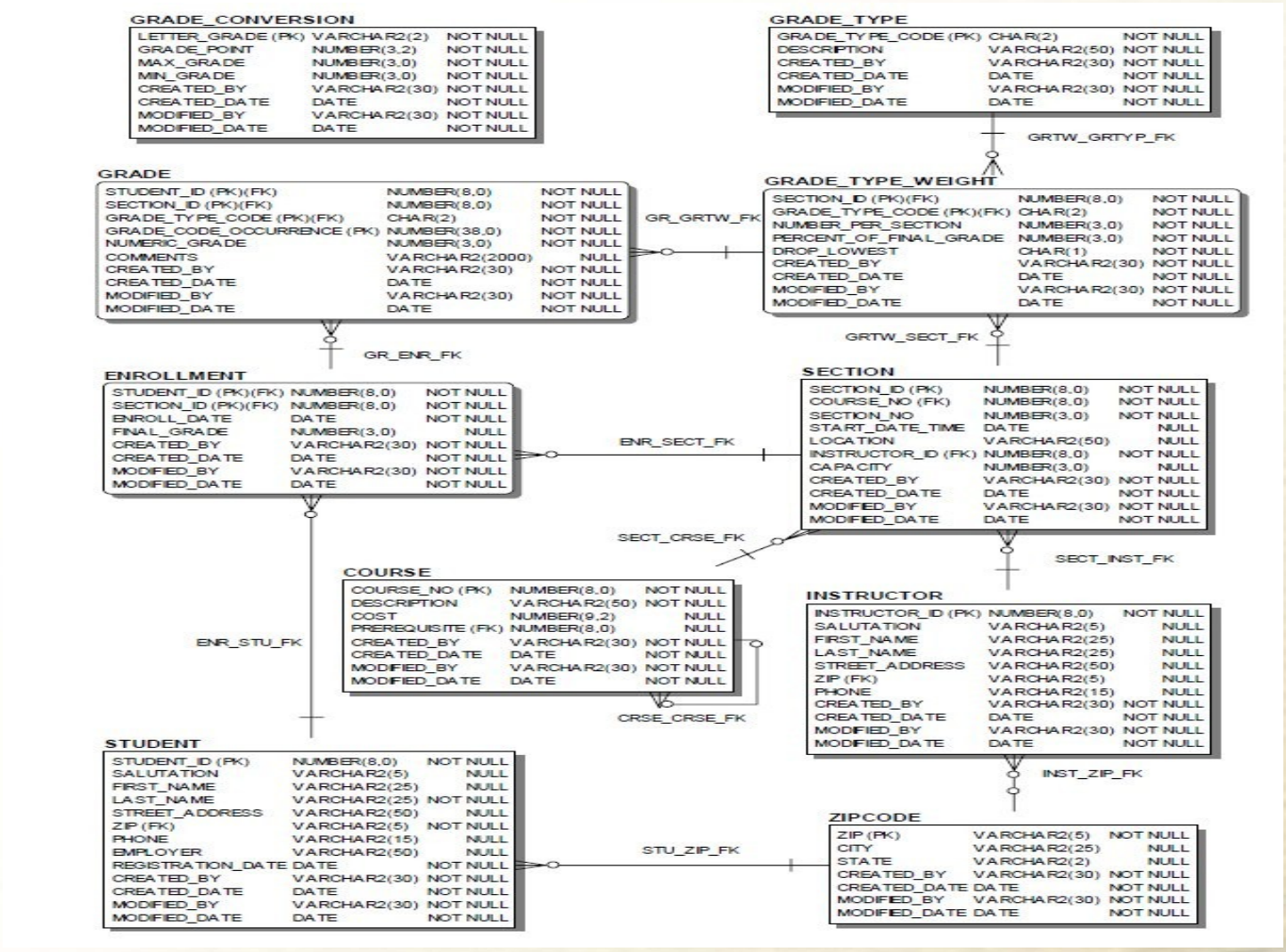
PLSQL

José Antonio Coria Fernández
Email: diplomadobd@gmail.com

Contenido

- Esquema base datos ejemplo
- Cursores Avanzados
- Colecciones y registros
- Funciones
- Manejo de Errores

Esquema base de datos ejemplo



Cursores avanzados

- Cursores con parametros
  - Hace un cursor más rehusable.
  - Es posible asignar valores predeterminados a los parámetros del cursor
  - El alcance de los parámetros del cursor es local al cursor
  - El modo de los parámetros sólo puede ser IN.
  - Cuando se ha declarado el cursor con parámetro, este debe ser llamado con el valor para el parámetro.

Cursores avanzados

- Ejemplo cursores con parametros

```
SET SERVEROUTPUT ON
SET ECHO ON
DECLARE
  CURSOR c_cp (p_estado IN zipcode.state%TYPE)
  IS
  SELECT zip, city, state
  FROM zipcode
  WHERE state = p_estado;
BEGIN
 FOR r_zip IN c_cp('NJ') LOOP
    DBMS_OUTPUT.PUT_LINE(r_zip.city||' '||r_zip.zip);
  END LOOP;
END;
/
```

Cursores avanzados

- En el ejemplo anterior, se mostro una forma alternativa para menejar los cursores. El cursor es llamado con un FOR Loop debido a la sintaxis simplificada que utiliza.

  Con el cursor Foor Loop, el proceso de apertura, recuperación y cerrado es manejado de forma implicita.

- Esto facilita el mantenimiento y codificación del bloque.
- Se emplea el FOR loop si se requiere recuperar y procesar cada registro del cursor hasta que se termine el proceso y salga del ciclo.

Cursores avanzados

Cursores anidados

- Permiten el recorrido a través de los datos en varios escenarios.

Por ejemplo en el cursor anterior podría recorrer los códigos postales, cuando un código postal X es encontrado, podría entonces buscar a los estudiantes que se encuentran en ese código postal se entonces, a través de otro cursor.

Cursores avanzados


```
DECLARE
    v_zip zipcode.zip%TYPE;
    v_student_flag CHAR;
    CURSOR c_zip IS
      SELECT zip, city, state
       FROM zipcode
      WHERE state = 'CT';
    CURSOR c_student IS
      SELECT first_name, last_name
       FROM student
      WHERE zip = v_zip;
  BEGIN
    FOR r_zip IN c_zip
    LOOP
      v_student_flag := 'N';
      v_zip := r_zip.zip;
      DBMS_OUTPUT.PUT_LINE(CHR(10));

   DBMS_OUTPUT.PUT_LINE('El estudiante vive en '||  r_zip.city);
```

Cursores avanzados


```
    FOR r_student in c_student
    LOOP
      DBMS_OUTPUT.PUT_LINE( r_student.first_name||' '||r_student.last_name);
      v_student_flag := 'Y';
    END LOOP;
    IF v_student_flag = 'N'
      THEN
      DBMS_OUTPUT.PUT_LINE
        ('No existen estudiantes para ese código postal');
    END IF;
  END LOOP;
END;
/
```

Cursores avanzados


```
SET SERVEROUTPUT ON
DECLARE
  v_amount course.cost%TYPE;
  v_instructor_id  instructor.instructor_id%TYPE;
  CURSOR c_inst IS
    SELECT first_name, last_name, instructor_id
     FROM instructor;
  CURSOR c_cost IS
    SELECT c.cost
     FROM course c, section s, enrollment e
    WHERE s.instructor_id = v_instructor_id
     AND c.course_no = s.course_no
     AND s.section_id = e.section_id;
BEGIN
  FOR r_inst IN c_inst
  LOOP
    v_instructor_id := r_inst.instructor_id;
    v_amount := 0;
```

Cursores avanzados


```
      DBMS_OUTPUT.PUT_LINE( 'Cantidad generada por instructor '||
       r_inst.first_name||' '||r_inst.last_name ||' es');
     FOR r_cost IN c_cost
     LOOP
       v_amount := v_amount + NVL(r_cost.cost, 0);
     END LOOP;
     DBMS_OUTPUT.PUT_LINE ('     '||TO_CHAR(v_amount,'$999,999'));
  END LOOP;
END;
/
```

Cursores avanzados

Práctica 2

* Cree un cursor que reciba como parámetro el nombre de la ciudad y nos de como salida el estado y el código postal.
* Cree un cursor anidado en el cual el cursor padre obtenga la información acerca de cada sección de un curso. El cursor hijo debe contar la matricula. La salida debe ser una sola línea para cada curso, con el nombre del curso, el número de sección y el total de matriculados.
* Cree un cursor anidado, que de como salida el nombre del estudiante seguido del curso o cursos que esta tomando, el promedio de calificación que obtuvo en cada uno de las evaluaciones que se llevaron a cabo. Ejemplo:

  Alumno: Jonathan Jaele

  Calificaciones del Curso: Intro to the Internet

  84.00 Final

  77.00 Midterm

Cursores FOR UPDATE y WHERE CURRENT

- El cursor FOR UPDATE se emplea únicamente cuando se quiere actualizar tablas en la base de datos.
- El propósito de utilizar esta cláusula (FOR UPDATE) es bloquear los registros de la tabla que se quiere actualizar y liberar el bloqueo.
- La sentencia COMMIT o ROLLBACK, liberan el bloqueo

Cursores FOR UPDATE y WHERE CURRENT

- Cuando se abre un cursor, los registros que cumplen con el criterio son identificados como parte de un conjunto activo. FOR UPDATE bloquea, estos registros.
- Si esta cláusula se emplea, los registros no serán regresados por el cursor hasta que una cláusula COMMIT haya sido emitido.

Cursores FOR UPDATE y WHERE CURRENT


```
DECLARE
CURSOR c_curso IS
SELECT course_no, cost
FROM course FOR UPDATE;
BEGIN
FOR r_curso IN c_curso
LOOP
IF r_curso.cost < 2500
THEN
UPDATE course
SET cost = r_curso.cost + 10
WHERE course_no = r_curso.course_no;
END IF;
END LOOP;
COMMIT;
END;
```

Cursores FOR UPDATE y WHERE CURRENT

En el siguiente código, antes de ejecutarlo mencione donde debería ponerse la sentencia commit y porque. Envíe su respuesta por correo.

```
DECLARE
  CURSOR c_grade(
    i_student_id IN enrollment.student_id%TYPE,
    i_section_id IN enrollment.section_id%TYPE)
  IS
    SELECT final_grade
     FROM enrollment
     WHERE student_id = i_student_id
      AND section_id = i_section_id
     FOR UPDATE;
  CURSOR c_enrollment IS
    SELECT e.student_id, e.section_id
     FROM enrollment e, section s
     WHERE s.course_no = 135
      AND e.section_id = s.section_id;
BEGIN
  FOR r_enroll IN c_enrollment
```

Cursores FOR UPDATE y WHERE CURRENT


```
  LOOP
    FOR r_grade IN c_grade(r_enroll.student_id,
                           r_enroll.section_id)
    LOOP
      UPDATE enrollment
        SET final_grade  = 90
       WHERE student_id = r_enroll.student_id
        AND section_id = r_enroll.section_id;
    END LOOP;
  END LOOP;
END;
/
```

Ejecute el código con el commit ubicado en la sección que considero pertinente.
¿que ocurre?

Cursores FOR UPDATE y WHERE CURRENT

- Se emplea el WHERE CURRENT OF cuando se requiere actualizar los registros recuperados recientemente.
- El WHERE CURRENT OF puede ser utilizado solamente con cursores FOR UPDATE OF.
- La *ventaja del WHERE CURRENT OF* es que brinda la posibilidad de eliminar la cláusula where de la sentencia UPDATE.

Ejemplo Cursores FOR UPDATE y WHERE CURRENT


```
DECLARE
CURSOR c_stud_zip IS
SELECT s.student_id, z.city
FROM student s, zipcode z
WHERE z.city = 'Brooklyn'
AND s.zip = z.zip
FOR UPDATE OF phone;
BEGIN
FOR r_stud_zip IN c_stud_zip
LOOP
DBMS_OUTPUT.PUT_LINE(r_stud_zip.student_id);
UPDATE student
SET phone = '718'||SUBSTR(phone,4)
WHERE CURRENT OF c_stud_zip;
END LOOP;
END;
```

Cursores FOR UPDATE y WHERE CURRENT

Capture el código anterior y ejecutelo. Posteriormente modifique el código, cambiando la línea donde se encuentra el

WHERE CURRENT OF c_stud_zip, por

WHERE student_id = r_stud_zip.student_id;

Y ejecutelo nuevamente

¿que ocurre?

¿cual cree que sería la diferencia de sustancial entre el where en el UPDATE y el WHERE CURRENT?

Colecciones y registros

* Una colección es un grupo de elementos del mismo tipo y son llamados elementos. Se pueden acceder a los elementos via los subíndices únicos(que empiezan en 1). Las listas y los arreglos son un ejemplo de colecciones.

- PL/SQL tiene tres tipos de colecciones:

  Arreglo asociativo(es previsto para almacenamiento temporal de datos)

  Tablas anidadas (pueden ser almacenada en una columna de una base de datos)

  Arreglos de tamaño variable

* Un registros, esta compuesto de diferentes tipos de datos y son llamados campos. Puede manejar la fila de una tabla, o algunas columnas de la fila de la tabla. Cada campo del registro corresponde a una columna de la tabla y se puede acceder a cada campo por su nombre.

Colecciones y registros

- Ejemplo 1 Arreglo asociativo

```
DECLARE
  CURSOR c_nombre IS
    SELECT last_name
      FROM student
     WHERE rownum <= 10;
  TYPE tipo_apellido IS TABLE OF student.last_name%TYPE
    INDEX BY BINARY_INTEGER; --cualquier tipo de datos  PL/SQL con algunas
restricciones --
 tabla_apellido tipo_apellido;
  v_contador INTEGER := 0;
BEGIN
  FOR reg_nombre  IN c_nombre LOOP
    v_contador := v_contador + 1;
    tabla_apellido(v_contador) := reg_nombre.last_name;
    DBMS_OUTPUT.PUT_LINE ('apellido('||v_contador||'): '||
      tabla_apellido(v_contador));
  END LOOP;
END;
```

Colecciones y registros

- Ejemplo 2. Arreglo asociativo

```
SET SERVEROUTPUT ON
DECLARE
TYPE poblacion IS TABLE OF NUMBER
INDEX BY VARCHAR2(64);
poblacion_ciudad poblacion;
i VARCHAR2(64);
BEGIN
poblacion_ciudad('Durango') := 2000;
poblacion_ciudad('Acapulco') := 750000;
poblacion_ciudad('DF') := 1000000;
poblacion_ciudad('Tlaxcala') := 2001;
i := poblacion_ciudad.FIRST;
WHILE i IS NOT NULL LOOP
 DBMS_Output.PUT_LINE ('Poblacion de ' || i || ' es de  ' || TO_CHAR(poblacion_ciudad(i)));
 i := poblacion_ciudad.NEXT(i);
 END LOOP;
 END;
 /
```

Colecciones y registros

- Ejemplo tabla anidada (1/2)

```
set serveroutput on
DECLARE
  CURSOR c_nombre IS
    SELECT last_name
      FROM student
    WHERE rownum <= 10;
  TYPE tipo_apellido IS TABLE OF student.last_name%TYPE;
**   tabla_apellido tipo_apellido;       --- := tipo_apellido();  --
  v_contador INTEGER := 0;
```

** Primero ejecute el código con esta línea, sin la cadena que esta entre --,

Posteriormente, vuelva a ejecutar el codigo haciendo la igualación. ¿Que ocurrió en ambos casos?, ¿Cual es la razón por la que se generan ambos resultados?

Colecciones y registros

- Ejemplo tabla anidada (2/2)

```
BEGIN
  FOR reg_nombre IN c_nombre LOOP
    v_contador := v_contador + 1;
    tabla_apellido.EXTEND;
    tabla_apellido(v_contador) := reg_nombre.last_name;
    DBMS_OUTPUT.PUT_LINE ('Apellido('||v_contador||'): '||
      tabla_apellido(v_contador));
  END LOOP;
END;
/
```

Métodos en colecciones

La siguiente lista explica los métodos de las colecciones que pemiten manipular u obtener información acerca de una colección:

- EXISTS regresa TRUE si un elemento específico existe en una colección. Puede utilizarse este método para evitar excepciones del tipo SUSBSCRIPT_OUTSIDE_LIMIT.
- COUNT regresa el total de elementos en una colección.
- EXTEND incrementa el tamaño de una colección.
- DELETE borra todos los elementos, los elementos de un rango o un elemento en partícular de una colección. NOTA: PL/SQL mantiene marcadores de posición de los elementos borrados.
- FIRST y LAST regresa los subíndices del primer y último elemento de una colección. Si el primer elemento de una tabla anidada es borrado, el método FIRST regresa un valor mayor a 1. Si el elemento ha sido borrado de la mitad de una tabla anidada, el método LAST regresa un valor mayor a que el método COUNT

Métodos en colecciones

- PRIOR y NEXT regresan el subíndice que antecede y sucede en una colección específica.
- TRIM elimina uno o un número específico de elementos del final de una colección.

NOTA: PL/SQL no mantiene un marcador de los elementos eliminados con TRIM

NOTA: Los métodos EXTEND y TRIM no pueden ser empleados con tablas por índice.

Métodos en colecciones

- Ejemplo: (1/3)

```
set serveroutput on
DECLARE
  TYPE indice_por_tipo IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
   indice_por_tabla indice_por_tipo;
  TYPE tipo_anidado IS TABLE OF NUMBER;
   tabla_anidada tipo_anidado := tipo_anidado(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
BEGIN
  FOR i IN 1..10 LOOP
    indice_por_tabla(i) := i;
 END LOOP;
  IF indice_por_tabla.EXISTS(3) THEN
    DBMS_OUTPUT.PUT_LINE ('indice_por_tabla(3) = '||indice_por_tabla(3));
  END IF;
```

Métodos en colecciones

- Ejemplo: (2/3)

```
tabla_anidada.DELETE(10);
tabla_anidada.DELETE(1,3);
indice_por_tabla.DELETE(10);
DBMS_OUTPUT.PUT_LINE ('tabla_anidada.COUNT = '||tabla_anidada.COUNT);
DBMS_OUTPUT.PUT_LINE ('indice_por_tabla.COUNT = '|| indice_por_tabla.COUNT);
DBMS_OUTPUT.PUT_LINE ('tabla_anidada.FIRST = '||tabla_anidada.FIRST);
DBMS_OUTPUT.PUT_LINE ('tabla_anidada.LAST = '||tabla_anidada.LAST);
DBMS_OUTPUT.PUT_LINE ('indice_por_tabla.FIRST = '||indice_por_tabla.FIRST);
DBMS_OUTPUT.PUT_LINE ('indice_por_tabla.LAST = '|| indice_por_tabla.LAST);
DBMS_OUTPUT.PUT_LINE ('tabla_anidada.PRIOR(2) = '|| tabla_anidada. PRIOR(2));
DBMS_OUTPUT.PUT_LINE ('tabla_anidada.NEXT(2) = '|| tabla_anidada.NEXT(2));
DBMS_OUTPUT.PUT_LINE ('indice_por_tabla.PRIOR(2) = '|| indice_por_tabla.PRIOR(2));
DBMS_OUTPUT.PUT_LINE ('indice_por_tabla.NEXT(2) = '|| indice_por_tabla.NEXT(2));
```

Métodos en colecciones

- Ejemplo: (2/3)

```
    tabla_anidada.TRIM(2);
  tabla_anidada.TRIM;
  DBMS_OUTPUT.PUT_LINE('tabla_anidada.LAST = '||tabla_anidada.LAST);
END;
/
```

•Del código anterior, de la explicación línea a línea de lo que ocurre.

Métodos en colecciones

- Práctica 3. Del siguiente código:

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR course_cur IS
    SELECT description
     FROM course;
    TYPE course_type IS TABLE OF course.description%TYPE
    INDEX BY BINARY_INTEGER;
  course_tab course_type;
  v_counter INTEGER := 0;
BEGIN
  FOR course_rec IN course_cur LOOP
    v_counter := v_counter + 1;
    course_tab(v_counter):= course_rec.description;
  END LOOP;
   FOR i IN 1..v_counter LOOP
    DBMS_OUTPUT.PUT_LINE('course('||i||'): '||course_tab(i));
  END LOOP;
END;
```

Métodos en colecciones

- Modifique el código para que solamente la primera y ultima fila del arreglo asociativo sea desplegado en la pantalla
- Desplegar el total de elementos del arreglo asociativo en pantalla
- Borre el último elemento, y despliegue el total de elementos del arreglo asociativo nuevamente.
- Borre el quinto elemento, y despliegue el total de número de elementos y el subíndice de el último elemento en el arreglo asociativo de nueva cuenta.

Métodos en colecciones

- Del siguiente código:

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR course_cur IS
    SELECT description
     FROM course;
  TYPE course_type IS TABLE OF course.description%TYPE;
  course_tab course_type := course_type();
  v_counter INTEGER := 0;
BEGIN
  FOR course_rec IN course_cur LOOP
    v_counter := v_counter + 1;
    course_tab.EXTEND;
    course_tab(v_counter) := course_rec.description;
  END LOOP;
END;
```

Métodos en colecciones

Del código anterior :

- borre el último elemento de la tabla anidada, y posteriormente asignele un nuevo valor. Vuelva a ejecutar el código.
- Elimine (TRIM) el último elemento de la tabla anidada, y asignele un nuevo valor. Ejecute de nuevo el código

Arreglos de tamaño variable

Los arreglos de tamaño variable VARRAY son como los arreglos en el sentido de los lenguajes de programación, debido a que tienen un tamaño fijo y usan un índice secuencial númerico.

Sintaxis:

```
TYPE nombre_tipo IS {VARRAY | VARYING ARRAY} (tamaño limite)
OF    tipo_elemento [NOT NULL];

varray_nombre NOMBRE_TIPO;
```

Arreglos de tamaño variable

EJEMPLO VARRAY (1/2)

```
DECLARE
TYPE integer_varray IS VARRAY(3) OF INTEGER;
varray_integer INTEGER_VARRAY := integer_varray(NULL,NULL,NULL);
BEGIN
dbms_output.put (CHR(10));
dbms_output.put_line('Varray inicializado con  NULLS.');
dbms_output.put_line('------------------------------');
FOR i IN 1..3 LOOP
dbms_output.put ('Varray de enteros ['||i||'] ');
dbms_output.put_line('['||varray_integer(i)||']');
END LOOP;
```

Arreglos de tamaño variable

EJEMPLO VARRAY (2/2)

```
varray_integer(1) := 11;
varray_integer(2) := 12;
varray_integer(3) := 13;
dbms_output.put (CHR(10));
dbms_output.put_line('Varray inicializado con valores.');
dbms_output.put_line('-------------------------------');
FOR i IN 1..3 LOOP
dbms_output.put_line('Varray de enteros ['||i||'] '|| '['||varray_integer(i)||']');
END LOOP;
END;
/
```

Arreglos de tamaño variable

EJEMPLO 2 VARRAY (NOTA: no se puede emplear, el método DELETE con VARRAY)

```
DECLARE
CURSOR c_nombre IS
SELECT last_name
FROM student
WHERE rownum <= 10;
TYPE tipo_apellido IS VARRAY(10) OF student.last_name%TYPE;
varray_apellido  tipo_apellido := tipo_apellido();
v_contador INTEGER := 0;
BEGIN
FOR reg_nombre IN c_nombre LOOP
v_contador := v_contador + 1;
varray_apellido.EXTEND;
varray_apellido(v_contador) := reg_nombre.last_name;
DBMS_OUTPUT.PUT_LINE ('Apellido('||v_contador||'): '|| varray_apellido(v_contador));
END LOOP;
```

Arreglos de tamaño variable

Práctica 4. Del siguiente código, encuentre los errores y corrijalos.

```
DECLARE
CURSOR c_ciudad IS
SELECT city
FROM zipcode
WHERE rownum <= 10;
TYPE tipo_ciudad IS VARRAY(10) OF zipcode.city%TYPE;
varray_ciudad tipo_ciudad;
v_contador INTEGER := 0;
BEGIN
FOR reg_ciudad IN c_ciudad LOOP
v_contador := v_contador + 1;
varray_ciudad(v_contador) := reg_ciudad.city;
DBMS_OUTPUT.PUT_LINE('Varray_ciudad('||v_contador||'): '||
varray_ciudad(v_contador));
END LOOP;
END;
```

Registros

- Un registro es una estructura que es similar a una fila de una tabla de una base de datos. Cada pieza de datos es almacenada en un campo con su propio nombre y tipo de datos.

- El atributo %ROWTYPE permite crear registros basados en tablas y basados en cursores. Es similar al atributo %TYPE el cual es empleado para definir variables escalares.

Registros

Ejemplo registro con tabla:

```
DECLARE
rec_curso%ROWTYPE;
BEGIN
SELECT *
INTO rec_curso
FROM course
WHERE course_no = 25;
DBMS_OUTPUT.PUT_LINE ('Curso No: '||rec_curso.course_no);
DBMS_OUTPUT.PUT_LINE ('Descripción del Curso: '||rec_curso.description);
DBMS_OUTPUT.PUT_LINE ('Prerequisito: '||rec_curso.prerequisite);
END;
```

Registros

En el código anterior, el registro rec_curso tiene la misma estructura de una fila de la tabla COURSE. Como resultado, no es necesario referenciar de forma individual a los campos del registro, en la cláusula SELECT INTO.

- Note que un registro no tiene un valor por si mismo, en lugar de esto, cada campo individual maneja un valor. De ahi que sea necesario para desplegar información del registro, se referencien campos individuales empleando un punto en la notación.

Registros

Ejemplo registro con cursor:

```
DECLARE
CURSOR c_estudiante IS
SELECT first_name, last_name, registration_date
FROM student
WHERE rownum <= 4;
reg_estudiante c_estudiante%ROWTYPE;
BEGIN
OPEN c_estudiante;
LOOP
FETCH c_estudiante  INTO reg_estudiante;
EXIT WHEN c_estudiante%NOTFOUND;
DBMS_OUTPUT.PUT_LINE ('Nombre: '|| reg_estudiante.first_name||' '||reg_estudiante.last_name);
DBMS_OUTPUT.PUT_LINE ('Fecha de Registro: '|| reg_estudiante.registration_date);
END LOOP;
END;
```

Registros

El registro reg_estudiante tiene la misma estructura, como la fila regresada por el cursor c_estudiante.

El resultado es similar al ejemplo de registro con tabla,

no es necesario referenciar campos individuales cuando los datos son recuperados (FETCH) del cursor al registro.

Registros

Del siguiente código, encuentre el error y mencione porque ocurrió.

```
DECLARE
reg_estudiante c_estudiante%ROWTYPE;
CURSOR c_estudiante IS
SELECT first_name, last_name, registration_date
FROM student
WHERE rownum <= 4;
BEGIN
OPEN c_estudiante;
LOOP
FETCH c_estudiante  INTO reg_estudiante;
EXIT WHEN c_estudiante%NOTFOUND;
DBMS_OUTPUT.PUT_LINE ('Nombre: '|| reg_estudiante.first_name||' '||reg_estudiante.last_name);
DBMS_OUTPUT.PUT_LINE ('Fecha de Registro: '|| reg_estudiante.registration_date);
END LOOP;
END;
```

Registros

Práctica 5. Modifique el siguiente código, y en lugar de proporcionar el valor del código postal en tiempo de ejecución, llene este valor empleando un cursor FOR Loop. La sentencia SELECT asociada al cursor debe regresar códigos postales que tienen más de un estudiante asociado. (1/2)

```
SET SERVEROUTPUT ON
DECLARE
TYPE last_name_type IS TABLE OF student.last_name%TYPE
INDEX BY BINARY_INTEGER;
TYPE zip_info_type IS RECORD
(zip VARCHAR2(5),
last_name_tab last_name_type);
CURSOR name_cur (p_zip VARCHAR2) IS
SELECT last_name
FROM student
WHERE zip = p_zip;
```

Registros

(2/2)

```
zip_info_rec zip_info_type;
v_zip VARCHAR2(5) := '&sv_zip';
v_counter INTEGER := 0;
BEGIN
zip_info_rec.zip := v_zip;
DBMS_OUTPUT.PUT_LINE ('Zip: '||zip_info_rec.zip);
FOR name_rec IN name_cur (v_zip) LOOP
v_counter := v_counter + 1;
zip_info_rec.last_name_tab(v_counter) := name_rec.last_name;
DBMS_OUTPUT.PUT_LINE ('Names('||v_counter||'): '||
zip_info_rec.last_name_tab(v_counter));
END LOOP;
END;
```

Al ejecutar y modificar este código, ¿que notó?, ¿podría emplear el término anidado?

Funciones

- Las funciones son otro tipo de código almacenado y son muy similares a los procedimientos.
- La principal diferencia es que una función es un bloque de PL/SQL que regresa un único valor.
- Pueden aceptar uno o varios parámetros o ningúno, pero debe debe tener una cláusula RETURN.
- Se tienen funciones pasadas por valor o por referencia.
- Los valores de salida puede ser cualquier tipo de datos de SQL o PL/SQL.
- Aún cuando no reciba un parametro debe regresar un valor

Funciones

- El tipo de datos a regresar debe ser declarado en la cabecera de la función.
- No es independiente de la forma que sería un procedimiento; debe emplearse en algún contexto.

SINTAXIS:

```
CREATE [OR REPLACE] FUNCTION nombre_funcion
(lista de parámetros)
RETURN TipoDeDatos
IS
BEGIN
<cuerpo>
RETURN (valor a regresar);
END;
```

Funciones

- Ejemplo:

```
CREATE OR REPLACE FUNCTION muestra_descripcion
   (i_numcurso course.course_no%TYPE)
RETURN varchar2
AS
v_descripcion varchar2(50);
BEGIN
 SELECT description
 INTO   v_descripcion
 FROM course
WHERE course_no = i_numcurso;
RETURN v_descripcion;
 EXCEPTION
 WHEN NO_DATA_FOUND
 THEN
```

Funciones

- Ejemplo:

```
 RETURN('El curso no esta en la base de datos');
  WHEN OTHERS
  THEN
  RETURN('Error al ejecutar la funcion muestra descripcion');
END;
```

- ------- -------------------- -------------------- -------------------------

```
SET SERVEROUTPUT ON
DECLARE
v_descripcion  VARCHAR2(50);
BEGIN
dbms_output.put (CHR(10));
v_descripcion:= muestra_descripcion(350);
dbms_output.put_line('El curso buscado es:'||v_descripcion);
dbms_output.put (CHR(10));
END;
```

Funciones

- Ejemplo 2: (1/2)

```
declare
 v_pi NUMBER:=3.14;
 function f_obtieneDiferencia(i_rad1 NUMBER,i_rad2 NUMBER)
 return NUMBER is
     v_area1 NUMBER;
     v_area2 NUMBER;
     v_salida NUMBER;
  function f_obtieneArea (i_rad NUMBER)
  return NUMBER
  is
  begin
    return v_pi*(i_rad**2);
  end;
```

Funciones

- Ejemplo 2: (2/2)

```
 begin
    v_area1 := f_obtieneArea (i_rad1);
    v_area2 := f_obtieneArea (i_rad2);
    v_salida :=v_area1-v_area2;
    return v_salida;
 end;
begin
 DBMS_OUTPUT.put_line ('Diferencia entre 3 y 4: '||f_obtieneDiferencia(4,3));
 DBMS_OUTPUT.put_line ('Diferencia entre 4 y 5: '||f_obtieneDiferencia(5,4));
 DBMS_OUTPUT.put_line ('Diferencia entre 5 y 6: '||f_obtieneDiferencia(6,5));
end;
/
```

Funciones

- Ejemplo 3:

```
DECLARE
v_idcurso NUMBER:=350;
v_costo NUMBER;
v_descripcion VARCHAR(50);
BEGIN
select cost,muestra_descripcion(v_idcurso)
into   v_costo,v_descripcion
from  course
where course_no=v_idcurso;
dbms_output.put (CHR(10));
DBMS_OUTPUT.put_line ('Curso: '||v_descripcion||' '||'Costo:'||v_costo);
dbms_output.put (CHR(10));
END
```

Funciones

PRÁCTICA 6.

- Cree una función que para un instructor dado, determine a cuantas secciones esta enseñando. Si el número es mayor o igual a 3, regrese un mensaje notificando al instructor que necesita vacaciones. De otra forma regrese un mensaje el número de secciones que el instructor esta enseñando.
- Cree una función que tome de entrada un zipcode.zip%TYPE y regrese un valor Booleano. La función regresará VERDADERO si el códio postal envíado no existe. Y FALSO cuando el código postal exista.

Manejo de errores

Dos tipos de errores pueden encontrarse en un programa: errores de compilación y errores en tiempo de ejecución.

PL/SQL tiene dos tipos de excepciones: definidas por el usuario e incorporadas.

Ejemplo (copie y ejecute el siguiente código):

```
DECLARE
v_num1 INTEGER := &sv_num1;
v_num2 INTEGER := &sv_num2;
v_result NUMBER;
BEGIN
v_result = v_num1 / v_num2;
DBMS_OUTPUT.PUT_LINE ('v_result: '||v_result);
END;
```

Manejo de errores

Al ejecutar el código anterior nos mostraría una salida como la siguiente:

ERROR at line 6:

ORA-06550: line 6, column 10:

PLS-00103: Encountered the symbol "=" when expecting one of the following:

:= . (@ % ;

The symbol ":= was inserted before "=" to continue.

- Como se muestra en las líneas anteriores, este es un error de de sintaxis.
- Al corregir v_result = v_num1 / v_num2; por v_result: = v_num1 / v_num2;
- Se elimina el mensaje de error, pero si se ingresan los valores 4 y 0, respectivamente se produce la siguiente salida:

Manejo de errores


```
Enter value for sv_num1: 4
old 2: v_num1 integer := &sv_num1;
new 2: v_num1 integer := 4;
Enter value for sv_num2: 0
old 3: v_num2 integer := &sv_num2;
new 3: v_num2 integer := 0;
DECLARE
*
ERROR at line 1:
ORA-01476: divisor is equal to zero
ORA-06512: at line 6
```

Aún cuando el ejemplo anterior no contiene error de sintaxis, termina prematuramente. Este ejemplo ilustra un error de tiempo de ejecución que el compilador no puede detectar.

Manejo de errores

Para manejar este tipo de errores en un programa, se debe adicionar un manejador de excepciones. El manejo de excepciones tiene la siguiente estructura:

EXCEPTION

WHEN NOMBRE_DE_EXCEPCION THEN

PROCESAMIENTO DE DECLARACIONES DE ERROR;

Ejemplo:

```
DECLARE
v_num1 INTEGER := &sv_num1;
v_num2 INTEGER := &sv_num2;
v_result NUMBER;
BEGIN
v_result := v_num1 / v_num2;
DBMS_OUTPUT.PUT_LINE ('v_result: '||v_result);
EXCEPTION
WHEN ZERO_DIVIDE THEN
DBMS_OUTPUT.PUT_LINE ('Un número no puede ser dividido por cero.');
END;
```

Manejo de errores

Nombre excepción	Ora error	Sql code	Ejecutado cuando...
ACCESS_INTO_NULL	06530	-6530	Un programa intenta asignar valoress a los atributos de objetos sin inicializar
CASE_NOT_FOUND	06592	-6592	Ningúna de las opciones en la cláusula WHEN de una declaración CASE es seleccionada, y no existe la cláusula else
COLLECTION_IS_NULL	06531	-6531	Un programa intenta asignar valores a los elementos sin inicializar de una tabla anidada o un arreglo variable.
CURSOR_ALREADY_OPEN	06511	-6511	Un programa intenta abrir un cursor ya abierto.

Manejo de errores

Nombre excepción	Ora error	Sql code	Ejecutado cuando...
DUP_VAL_ON_INDEX	00001	-1	Un programa intenta almacenar un valor duplicado en una columna con una restriccion de indice unico
INVALID_CURSOR	06592	-1001	Un programa itenta una operacion con cursor que no es permitida, tal como cerrar un cursor que no esta abierto
INVALID_NUMBER	06531IN	-1722	En una declaración SQL, la conversión de una cadena de caracteres a número falla porque la cadena no repersenta un numero válido
LOGIN_DENIED	06511	-1017	Un progama intenta registrase a una base de datos con un usuario o contraseña invalido.

Manejo de errores

Nombre excepción	Ora error	Sql code	Ejecutado cuando...
NO_DATA_FOUND	0143	+100	Una declaración select into no regresa una fila, o un programa hace referencia a un elemento eliminado de una tabla anidada
NOT_LOGGED_ON	01012	-1012	Un proceso de un programa trata de concectarse a una base de datos sin estar conectado
PROGRAM_ERROR	06501	-06501	PL/SQL tiene un problema interno
ROWTYPE_MISMATCH	06504	-06504	La varaiable de cursor involucrado en una asignación tiene un tipo de regreso incompatible
STORAGE_ERROR	06500	-6500	PL/SQL ejecuto fuera de memoria o la memoria fue corrompida

Manejo de errores

Nombre excepcion	Ora error	Sql code	Ejecutado cuando...
SUBSCRIPT_BEYOND_COUNT	06533	-6533	Un programa hace referencia a una elemento de un varray o tabla anidada utilizanod un indice más grande al tamaño de la colección
SUBSCRIPT_OUTSIDE_LIMIT	06532	-6532	Un programa hace referencia a un elemento de un varrray o tabla anidada usando un índice que esta fuera del rango legal
SYS_INVALID_ROWID	01410	-1410	La conversión de cadena de caracteres a un rowid falla porque la cadena no repreesenta un rowid válido
TIMEOUT_ON_RESOURCE	00051	-51	Tiempo excedido ocurrio mientras la base de datos está esperando por un recurso
TOO_MANY_ROWS	01422	-1422	Una declaración select into regresa más de una fila

Manejo de errores

Nombre excepcion	Ora error	Sql code	Ejecutado cuando...
VALUE_ERROR	06502	-6502	Ocurrio un error aritmetico, de conversión, al truncar o condición de tamaño.
ZERO_DIVIDED	01476	-1476	Un programa intenta dividir un número entre cero
SELF_IS_NULL	30625	-30625	Un programa intento invocar un método MEMBER, pero la instancia deel tipo de objeto no fue inicializada

Manejo de errores

EJEMPLO:

```
DECLARE
emp_column VARCHAR2(30) := 'last_name';
table_name VARCHAR2(30) := 'emp';
temp_var VARCHAR2(30);
BEGIN
temp_var := emp_column;
SELECT COLUMN_NAME INTO temp_var FROM USER_TAB_COLS
WHERE TABLE_NAME = 'EMPLOYEES'
AND COLUMN_NAME = UPPER(emp_column);
temp_var := table_name;
SELECT OBJECT_NAME INTO temp_var FROM USER_OBJECTS
WHERE OBJECT_NAME = UPPER(table_name)
AND OBJECT_TYPE = 'TABLE';
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No se encontraron Datos para  SELECT en ' || temp_var);
END;
/
```

Manejo de errores

PRÁCTICA 7:

Del siguiente código haga las modificación pertinente de tal forma que haga uso del manejo de exepciones definidas

```
SET SERVEROUTPUT ON;
DECLARE
v_numero NUMBER := &sv_numero;
BEGIN
IF v_numero >= 0 THEN
DBMS_OUTPUT.PUT_LINE ('La raiz cuadrada de '||v_numero||' es '||SQRT(v_numero));
ELSE
DBMS_OUTPUT.PUT_LINE ('Un numero no puede ser negativo');
END IF;
END;
/
```