Facultad de Estudios Superiores
Acatlán

PLSQL

José Antonio Coria Fernández
Email: diplomadobd@gmail.com

Contenido

- Manejo de Errores (definidos por el usuario)
- Disparadores
- Paquetes

Manejo de errores

- Excepciones definidas por el usuario
- Frecuentemente en los programas es requerido manejar un problema que es específico al programa.
- Por ejemplo si se quiere validar que el valor ingresado por un usuario sea un número entero no negativo, , para encontrar un alumno con un número de cuenta específico.

Manejo de errores

- Para realizar esta tarea, es necesario considerar lo siguiente:
- Las excepciones deben ser declarados solamente en la parte declarativa de un bloque de PL/SQL.
  - DECLARE
  - nombre_excepcion EXCEPTION;

NOTA: Una excepción no puede aparecer en una declaración de asignación o declaración SQL.

Manejo de errores

- Alcance de las excepciones
  - No se puede definir una excepción dos veces en el mismo bloque, sin embargo es posible declarar la misma excepción en dos bloques diferentes
  - Las excepciones declaradas en un bloque son consideradas locales a ese bloque y globales a todos sus sub-bloques.
  - Si se redeclara una excepción global en un subbloque, la declaración local prevalece

Manejo de errores

- Ejemplo: (1/2)

```
DECLARE
v_idestudiante student.student_id%type := &sv_idestudiante;
v_total_cursos NUMBER;
e_id_invalido EXCEPTION;
BEGIN
IF v_idestudiante < 0 THEN
RAISE e_id_invalido;
ELSE
SELECT COUNT(*)
INTO v_total_cursos
FROM enrollment WHERE student_id = v_idestudiante;
```

Manejo de errores

- Ejemplo: (2/2)

```
DBMS_OUTPUT.PUT_LINE ('El estudiante esta registrado en '|| v_total_cursos||' courses');
END IF;
DBMS_OUTPUT.PUT_LINE ('Ninguna excepcion ha sido encontrado');
EXCEPTION
WHEN e_id_invalido THEN DBMS_OUTPUT.PUT_LINE ('Un id no puede ser negativo');
END;
```

Manejo de errores

Es importante hacer notar que:

La declaración RAISE debe ser utilizado en conjunto con una declaración IF.

De otra manera, el control de la ejecución es transferido a la sección de manejo de excepciones del bloque para cada ejecución, aún cuando no se haya generado un error.

Manejo de errores

Ejemplo alcances de errores: (1/2)

```
DECLARE
e_fechdeb EXCEPTION;  --declaracion externa
v_numact NUMBER;
BEGIN
DECLARE ---------- empieza sub-bloque
e_fechdeb EXCEPTION; -- esta declaración (interna) prevalece
v_numact NUMBER;
fecha_debida DATE := SYSDATE - 1;
fecha_actual DATE := SYSDATE;
BEGIN
```

Manejo de errores

Ejemplo alcances de errores: (2/2)

```
IF fecha_debida < fecha_actual THEN
RAISE e_fechdeb; -- Este no se maneja
END IF;
END; ------------ termina sub-bloque
EXCEPTION
WHEN e_fechdeb THEN
DBMS_OUTPUT.PUT_LINE
('Manejo de la excepcion e_fechdeb.');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE
('No es posible reconocer la excepcion e_fechdeb en este alcance.');
END;
/
```

Manejo de errores

PRACTICA 8. Copie y ejecute el código anterior envíe sus respuestas por email. ¿Qué nos muestra de salida?

¿La linea WHEN e_fechdeb THEN es activada por la declaración de error externa o interna?

¿Que ocurre si al código anterior, le cambia el nombre a la declaración de error externa?

¿que ocurre si antes del end del subbloque agrega las siguientes líneas?

```
EXCEPTION
WHEN e_fechdeb THEN
DBMS_OUTPUT.PUT_LINE ('Interno: Manejo de la excepcion e_fechdeb.');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE ('No es posible reconocer la excepcion e_fechdeb en este alcance.');
```

Manejo de errores

PRACTICA 8. Copie y ejecute el código anterior envíe sus respuestas por email. ¿Qué nos muestra de salida?

¿La linea WHEN e_fechdeb THEN es activada por la declaración de error externa o interna?

¿Que ocurre si al código anterior, le cambia el nombre a la declaración de error externa?

¿que ocurre si antes del end del subbloque agrega las siguientes líneas?

```
EXCEPTION
WHEN e_fechdeb THEN
DBMS_OUTPUT.PUT_LINE ('Interno: Manejo de la excepcion e_fechdeb.');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE ('No es posible reconocer la excepcion e_fechdeb en este alcance.');
```

Manejo de errores

Propagación de excepciones

Cuando una excepción es iniciada(declaración RAISE) y si PL/SQL no puede encontrar un manedjo de la excepción el el bloque actual o subprograma, la exepción se propaga. Es decir, la exepción se reproduce así misma en los bloques externos hasta encontrar un manejador de excepciones o bien no haya más bloques en donde buscar.

Si no se encuentra un manejador de excepciones, PL/SQL regresa un error de excepción sin manejador a pantalla.

Manejo de errores

Ejemplo de propagación de excepciones:

```
SET SERVEROUTPUT ON
DECLARE
v_nombre_est VARCHAR2(15); ---linea 3
BEGIN
v_nombre_est := 'JOSE ANTONIO CORIA'; ---- linea 5
DBMS_OUTPUT.PUT_LINE ('Mi nombre es '||v_nombre_est);
DECLARE
v_nombre_din VARCHAR2(15);
BEGIN
v_nombre_din := '&sv_nombre_din';
DBMS_OUTPUT.PUT_LINE ('Tu nombre es '||v_nombre_din);
EXCEPTION
WHEN VALUE_ERROR THEN
```

Manejo de errores

Ejemplo de propagación de excepciones:

DBMS_OUTPUT.PUT_LINE ('El nombre es muy largo (interno');

END;

EXCEPTION

WHEN VALUE_ERROR THEN

DBMS_OUTPUT.PUT_LINE ('El nombre es muy largo (externo)');

END;

¿que ocurre si elimina la linea 5 y la linea 3 la cambia por v_nombre_est VARCHAR2(15) := 'jose antonio coria'?

¿porque el manejador de excepciones no se activa?

Manejo de errores

- En ocasiones se requiere manejar un error de oracle que tiene asociado un “número” en partícular, pero no así un nombre con el cual pueda ser referenciado.
- Para capturar este error, se puede emplear un constructor llamado PRAGMA.
- La instrucción PRAGMA es una instrucción especial del compilador PL/SQL y son procesadas en el tiempo de compilación.
- Nos permite asociar el número de error de oracle con el nombre de un manejador de error definido por el usuario.

Manejo de errores

- Ejemplo:

```
DECLARE
v_cp  zipcode.zip%type := '&sv_cp';
BEGIN
DELETE FROM zipcode
WHERE zip = v_cp;
DBMS_OUTPUT.PUT_LINE ('EL CP '||v_cp||' se ha borrado');
COMMIT;
END;
```

- Para este ejemplo ingrese el valor 06870, ¿que sucede?

Manejo de errores

Se mostrarían las siguientes líneas:

Enter value for sv_cp: 06870

old 2: v_cp zipcode.zip%type := '&sv_cp';

new 2: v_cp zipcode.zip%type := '06870';

DECLARE

*

ERROR at line 1:

ORA-02292: integrity constraint (STUDENT.STU_ZIP_FK) violated - child record found

ORA-06512: at line 4

Manejo de errores

Ahora retomando el error número **ORA-02292,** se podría efectuar la siguiente modificación al código anterior.

```
DECLARE
v_cp zipcode.zip%type := '&sv_cp';
e_existe_estudiante EXCEPTION;
PRAGMA EXCEPTION_INIT(e_existe_estudiante, -2292);
BEGIN
DELETE FROM zipcode
WHERE zip = v_cp;
DBMS_OUTPUT.PUT_LINE ('El CP  '||v_cp||' se ha borrado');
COMMIT;
EXCEPTION
WHEN e_existe_estudiante THEN
DBMS_OUTPUT.PUT_LINE ('Primero se debe borrar estudiantes para este  CP');
END;
```

Manejo de errores

Las excepciones definidas por el usuario de forma dinámica permiten manejar una excepción, asignarle un número, y administrar si se agrega o no el nuevo error a la lista de errores (conocido como un error de pila). La sintaxis es:

RAISE_APPLICATION_ERROR(numero_error, mensaje_error [, guarda_error])

El primer parámetro toma el número de error en el rango de -20,000 a -20,999. Se genera un error ORA-21000 si se proporciona algún otro valor

- El segundo parámetro es un mensaje de error.
- El último parámetro es opcional y su valor predeterminado es FALSE.

Manejo de errores

Ejemplo

```
DECLARE
v_idestudiante student.student_id%type := &sv_idestudiante;
v_total_cursos NUMBER;
BEGIN
IF v_idestudiante < 0 THEN
RAISE_APPLICATION_ERROR (-20000, 'Un id estudiante no puede ser negativo');
ELSE
SELECT COUNT(*)
INTO v_total_cursos
FROM enrollment
WHERE student_id = v_idestudiante;
DBMS_OUTPUT.PUT_LINE ('El estudiante esta registrado en: '||v_total_cursos||' cursos');
END IF;
END;
```

Manejo de errores

PRACTICA 9. Del siguiente código agregue un manejo de excepción definido por el usuario de forma dinámica cuando un curso cuente con más de 10 alumnos.

Diseñe otro código que maneje la excepción considerando únicamente el número de error. (PRAGMA) (1/2)

```
SET SERVEROUTPUT ON
DECLARE
v_estudiantes NUMBER(3) := 0;
BEGIN
SELECT COUNT(*)
INTO v_estudiantes
```

Manejo de errores

(2 / 2)

```
FROM enrollment e, section s
WHERE e.section_id = s.section_id
AND s.course_no = 25
AND s.section_id = 89;
DBMS_OUTPUT.PUT_LINE ('El Curso 25, sección 89 tiene '||v_estudiantes||
' estudiantes');
END;
```

DISPARADORES

Un disparador es un programa con nombre que es almacenado en la base de datos y define una acción en respuesta a un evento especifico.

Pueden utilizarse los disparadores para imponer ciertas restricciones de integridad referencial, para imponer restricciones empresariales complejas o para auditar los cambios realizados en los datos.

Los disparadores se basan en el modelo ECA (Event-Condition-Action, **Suceso-Condición-Acción**).

DISPARADORES

- Suceso: que dispara la regla especificada, puede ser:

- Una instrucción INSERT, UPDATE o DELETE sobre una tabla o vista especificada (DML)
- Una instrucción CREATE, ALTER o DROP ejecutada sobre cualquier objeto del esquema. (DDL)
- Un arranque de la base de datos o detención de la instancia, o bien el inicio o fin de una sesión por parte del usuario (LOGON, LOGOFF,STARTUP,SHUTDOWN)
- Un mensaje de error especifico o cualquier mensaje de error. (SERVERERROR)

DISPARADORES

- La condición: determina si la acción debe ejecutarse. La condición es opcional, pero si se especifica, la acción se ejecutara únicamente si la condición se evalúa como verdadera.
- La acción: contiene el código y las instrucciones SQL que hay que ejecutar cuando se produce un disparo y la condición de disparo se evalúo como verdadera.

DISPARADORES

Se tienen 5 tipos de disparadores:

- Disparadores de lenguaje de definición de datos. Se disparan cuando se crea, cambia o mueve un objeto del esquema de la base de datos.

- Disparadores de lenguaje de manipulación de datos. Se disparan cuando se inserta, actualiza o borran datos de una tabla. Se pueden disparar una vez que los cambios se realizaron en la tabla o para cada cambio de fila. Se suelen emplear para auditar, verificar, salvar y reemplazar valores que han sido cambiados.

DISPARADORES

- Disparadores compuestos. Estos disparadores actúan a nivel de fila y declaración, cuando se inserta, actualiza o borra un dato de la tabla. Permiten capturar información en 4 marcas de tiempo: antes de disparar la declaración, antes de que cada fila cambie desde la declaración disparada, después de que cada fila cambio desde la declaración disparada y después de disparar la declaración.

Se pueden utilizar para auditar, verificar, salvar y reemplazar valores antes de que sean cambiados cuando se necesita tomar una acción combinada.

DISPARADORES

- Disparadores en lugar de (instead of). Posibilitan el paro de la ejecución de una declaración DML y redireccionarla. Aplica reglas de negocio.

- Disparadores de eventos de la base de datos o sistema. Se activan cuando una actividad del sistema ocurre en la base de datos, como el inicio o cierre de sesión.

DISPARADORES

Sintaxis:

```
CREATE [OR REPLACE] TRIGGER Nombre_Disparador
{BEFORE|AFTER} Evento_Disparador ON Nombre_tabla
[FOR EACH ROW]
[FOLLOWS Otro_Disparador]
[ENABLE/DISABLE]
[WHEN condicion]

DECLARE
declaracion
BEGIN
ejecutar_sentencias
EXCEPTION
manejo_excepciones_sentencias
END;
```

DISPARADORES

Ejemplo 1:

```
CREATE OR REPLACE TRIGGER Imprime_cambio_salario
BEFORE DELETE OR INSERT OR UPDATE ON scott.emp
FOR EACH ROW
DECLARE
dif_sal number;
BEGIN
dif_sal := :NEW.SAL - :OLD.SAL;
dbms_output.put('Salario anterior: ' || :OLD.sal);
dbms_output.put('Nuevo Salario: ' || :NEW.sal);
dbms_output.put_line(' Diferencia: ' || dif_sal);
END;
/
```

```
UPDATE scott.emp SET sal = sal + 500.00 WHERE deptno = 10;
```

DISPARADORES

Ejemplo 1:

```
SQL> select sal from scott.emp where deptno=10;
      SAL
----------
      2450
      5000
      1300
SQL> UPDATE scott.emp SET sal = sal + 500.00 WHERE deptno = 10;
Salario anterior: 2450Nuevo Salario: 2950 Diferencia: 500
Salario anterior: 5000Nuevo Salario: 5500 Diferencia: 500
Salario anterior: 1300Nuevo Salario: 1800 Diferencia: 500
3 rows updated.

SQL>  select sal from scott.emp where deptno=10;
      SAL
----------
      2950
      5500
      1800
```

DISPARADORES

Ejemplo 2:

```
CREATE OR REPLACE TRIGGER  ai_estudiante
BEFORE INSERT ON scott.student2
FOR EACH ROW
DECLARE
v_id_estudiante  STUDENT.STUDENT_ID%TYPE;
BEGIN
SELECT STUDENT_ID_SEQ.NEXTVAL
INTO v_id_estudiante
FROM dual; /* Esta tabla es utilizada en oracle cuando se necesita ejecutar un comando SQL que no tiene lógicamente el nombre de tabla  */
:NEW.student_id := v_idestudiante;  /* :NEW pseudoregistro */
:NEW.created_by := USER;
:NEW.created_date := SYSDATE;
:NEW.modified_by := USER;
:NEW.modified_date := SYSDATE;
END;
```

DISPARADORES

Ejemplo 2:

INSERT INTO scott.student2 (first_name, last_name, zip, registration_date)
VALUES ('Jesus', 'Vazquez', '00914', SYSDATE);

(**student_id**,first_name,last_name,zip,registration_date,**created_by**,**created_date**,**modified_by**,**modified_date**) ¿porque no se agregan los campos que están en negritas?

Ejemplo 3:

Crear la tabla estadisticas con la siguiente estructura:

Name	Type
TABLE_NAME	VARCHAR2(30)
TRANSACTION_NAME	VARCHAR2(10)
TRANSACTION_USER	VARCHAR2(30)
TRANSACTION_DATE	DATE

DISPARADORES

Ejemplo 3:

```
CREATE OR REPLACE TRIGGER dad_instructor
AFTER UPDATE OR DELETE ON scott.INSTRUCTOR
DECLARE
v_tipo VARCHAR2(10);
BEGIN
IF UPDATING THEN
v_tipo := 'UPDATE';
ELSIF DELETING THEN
v_tipo := 'DELETE';
END IF;
UPDATE estadisticas
SET transaction_user = USER,
transaction_date = SYSDATE
WHERE table_name = 'INSTRUCTOR'
AND transaction_name = v_tipo;
IF SQL%NOTFOUND THEN /* se evalúa en Verdadero si la sentencia update no actualizó ningún registro*/
INSERT INTO estadisticas VALUES ('INSTRUCTOR', v_type, USER, SYSDATE);
END IF;
END;
```

DISPARADORES

Ejemplo 4:

```
CREATE OR REPLACE TRIGGER aia_instructor
BEFORE INSERT OR UPDATE OR DELETE ON scott.INSTRUCTOR
DECLARE
v_dia VARCHAR2(10);
BEGIN
v_ddia := RTRIM(TO_CHAR(SYSDATE, 'DAY'));
IF v_dia LIKE ('S%') THEN /* fecha en sabado o domingo, notación inglés*/
RAISE_APPLICATION_ERROR (-20000, 'una tabla no puede ser modificada durante horarios fuera de oficina');
END IF;
END;


UPDATE instructor
SET zip = 10025
WHERE zip = 10015;
```

DISPARADORES

Ejemplo 5: (disparadores aplicados a vistas) (½)

```
GRANT CREATE VIEW TO estudiante; /* se da permisos de crear vista si no es usuario SYS*/
1. Cree la vista de la tabla instructor
CREATE VIEW instructor_vista_resumen AS
SELECT i.instructor_id, COUNT(s.section_id) total_courses
FROM instructor i
LEFT OUTER JOIN section s  ON (i.instructor_id = s.instructor_id)
GROUP BY i.instructor_id;
2. Borre un registro de la vista
DELETE FROM instructor_vista_resumen
WHERE instructor_id = 109;
3. Agregue el siguiente disparador y vuelva a ejecutar el borrado del registro
CREATE OR REPLACE TRIGGER eld_instructor_resumen
INSTEAD OF DELETE ON instructor_vista_resumen
FOR EACH ROW
BEGIN
DELETE FROM instructor
WHERE instructor_id = :OLD.INSTRUCTOR_ID;
END;
```

DISPARADORES

Ejemplo 5: (disparadores aplicados a vistas) (2/2)

```
DELETE FROM instructor_vista_resumen
WHERE instructor_id = 109;
```

Para diseñar este tipo de disparadores, se debe estar consciente de dos factores importantes: la relación entre las tablas en la base de datos y el efecto dominó que un diseño en particular puede introducir.

DISPARADORES

Práctica 10:

```
CREATE OR REPLACE TRIGGER ai_curso
BEFORE INSERT ON COURSE
FOR EACH ROW
DECLARE
v_numcurso COURSE.COURSE_NO%TYPE;
BEGIN
SELECT COURSE_NO_SEQ.NEXTVAL
INTO v_numcurso
FROM DUAL;
:NEW.COURSE_NO := v_numcurso;
:NEW.CREATED_BY := USER;
:NEW.CREATED_DATE := SYSDATE;
:NEW.MODIFIED_BY := USER;
:NEW.MODIFIED_DATE := SYSDATE;
END;
```

Del código anterior conteste las siguientes preguntas:

1) ¿Que tipo de disparador es creado en la tabla COURSE?

2) ¿Cuando una declaración INSERT se emite en la tabla curso, que acción ejecuta el disparador?

3) Modificar este disparador de modo que si un curso de requisito previo es enviado en el momento de la inserción, su valor es cotejarse con los cursos existentes en la tabla CURSO

PAQUETES

Los paquetes son librerías almacenadas en la base de datos. Su propietario es el dueño del esquema donde son creados.

Los paquetes permiten agrupar funciones y procedimientos como componentes dentro de librerías. Dentro de estos paquetes de librerías se puede compartir, variables, tipos y componentes.

Un paquete contiene lo siguiente:

- Los métodos GET Y SET para las variables del paquete, si desea evitar que otros subprogramas lean y escribir directamente.

- Declaraciones de cursor con el texto de consultas SQL. La reutilización del texto de la consulta en múltiples ubicaciones es más rápido que escribir de nuevo la misma consulta cada vez. También es más fácil de mantener si tiene que cambiar una consulta que se utilizado en muchos lugares.

PAQUETES

- Declaraciones de excepciones. Es posible hacer referencia a estos desde diferentes subprogramas, de modo que es posible manejar las excepciones dentro de los programa invocados.

- Declaraciones de subprogramas que invocan unos a otros. No se requiere llamar en una secuencia de ejecución un grupo de subprogramas.

- Declaraciones de subprogramas sobrecargados. Se pueden crear múltiples variaciones de un subprograma, con los mismos nombres, pero diferentes conjuntos de parámetros.

PAQUETES

- Variables que se requieren tener disponibles entre las llamadas de subprograma en la misma sesión. Es posible tratar las variables en un paquete como variables globales.

- Declaraciones de tipos de colecciones PL/SQL. Para pasar una colección como un parámetro entre subprogramas almacenados, se debe declarar el tipo en un paquete de modo que ambos la invocación y el programa invocado puede referirse a ella.

PAQUETES

Ventajas de los paquetes:

- Modularidad
- Diseño de aplicaciones más fácil
- Ocultamiento de información
- Funcionalidad agregada
- Mejor Desempeño

PAQUETES

Especificación del paquete:

Contiene información acerca del contenido del paquete, pero no del código para los procedimientos y funciones.

También contiene la declaración de variables públicas y globales.

PACKAGE nombre_paquete
IS
[Declaración de variables y tipos]
[Especificación de cursores]
[Especifiación de módulos]
END [nombre_paquete];

PAQUETES

Cuerpo del paquete:

Contiene el actual código ejecutable para los objetos descritos en la especificación del paquete. Contiene el código para todos los procedimientos y funciones descritas en la especificación.

PACKAGE BODY nombre_paquete
IS
[Declaración de variables y tipos]
[Especificación y declaración SELECT de cursores]
[Especificación y cuerpo de los módulos]
[BEGIN
Declaraciones ejecutables]

PAQUETES

Ejemplo:

```
CREATE OR REPLACE PACKAGE maneja_estudiantes
 AS
 PROCEDURE encuentra_nombre_est
 (i_idestudiante IN student.student_id%TYPE,
 o_nombre OUT student.first_name%TYPE,
 o_apellido OUT student.last_name%TYPE
 );
 FUNCTION id_es_valido
 (i_idestudiante IN student.student_id%TYPE)
 RETURN BOOLEAN;
 END maneja_estudiantes;
```

PAQUETES


```
CREATE OR REPLACE PACKAGE BODY maneja_estudiantes
 AS
 PROCEDURE encuentra_nombre_est
 (i_idestudiante IN student.student_id%TYPE,
 o_nombre OUT student.first_name%TYPE,
 o_apellido OUT student.last_name%TYPE
 )
 IS
 v_idestudiante student.student_id%TYPE;
 BEGIN
 SELECT first_name, last_name
 INTO o_nombre, o_apellido
 FROM student
 WHERE student_id = i_idestudiante;
```

PAQUETES


```
EXCEPTION
 WHEN OTHERS
 THEN
 DBMS_OUTPUT.PUT_LINE
 ('Error para encontrar al id_estudiante: '||v_idestudiante);
 END encuentra_nombre_est;
 FUNCTION id_es_valido
 (i_idestudiante IN student.student_id%TYPE)
 RETURN BOOLEAN
IS
 v_idcontador number;
```

PAQUETES


```
BEGIN
 SELECT COUNT(*)
 INTO v_idcontador
 FROM student
 WHERE student_id = i_idestudiante;
 RETURN 1 = v_idcontador;
 EXCEPTION
 WHEN OTHERS
 THEN
 RETURN FALSE;
 END id_es_valido;
 END maneja_estudiantes;
```

PAQUETES


```
SET SERVEROUTPUT ON
DECLARE
v_nombre student.first_name%TYPE;
v_apellido  student.last_name%TYPE;
BEGIN
IF maneja_estudiantes.id_es_valido(&&v_id) /* se pide el valor y se define la variable como el valor.
Cualquier referencia subsecuente no se pedirá el valor nuevamente */
THEN
maneja_estudiantes.encuentra_nombre_est(&&v_id, v_nombre,
v_apellido);
DBMS_OUTPUT.PUT_LINE('Estudiante No. '||&&v_id||' es '
||v_apellido||', '||v_nombre);
ELSE
DBMS_OUTPUT.PUT_LINE
('El  ID Estudiante: '||&&v_id||' no esta en la base de datos.');
END IF;
END;
```

PAQUETES

Práctica 11. Del siguiente código, cree la especificación del paquete y la llamada al paquete almacenado.

```
1 CREATE OR REPLACE PACKAGE BODY pac_escuela AS
2 PROCEDURE descuento
3 IS
4 CURSOR c_descuentogrupo
5 IS
6 SELECT distinct s.course_no, c.description
7 FROM section s, enrollment e, course c
8 WHERE s.section_id = e.section_id
9 GROUP BY s.course_no, c.description,
10 e.section_id, s.section_id
11 HAVING COUNT(*) >=8;
12 BEGIN
13 FOR r_descuentogrupo IN c_descuentogrupo
14 LOOP
```

PAQUETES


```
15 UPDATE course
16 SET cost = cost * .95
17 WHERE course_no = r_descuentogrupo.course_no;
18 DBMS_OUTPUT.PUT_LINE ('Un 5% de descuento ha sido otorgado a' ||
r_descuentogrupo.course_no||''||r_descuentogrupo.description);
21 END LOOP;
22 END descuento;
23 FUNCTION id_nuevoinstructor
24 RETURN instructor.instructor_id%TYPE
25 IS
26 v_nuevoinstructor instructor.instructor_id%TYPE;
27 BEGIN
28 SELECT INSTRUCTOR_ID_SEQ.NEXTVAL
29 INTO v_nuevoinstructor
30 FROM dual;
31 RETURN v_nuevoinstructor;
32 EXCEPTION
```

PAQUETES


```
33 WHEN OTHERS
34 THEN
35 DECLARE
36 v_sqlerrm VARCHAR2(250) := SUBSTR(SQLERRM,1,250);
37 BEGIN
38  RAISE_APPLICATION_ERROR(-20003,'Error en el ID de INSTRUCTOR: '||
v_sqlerrm);
39 END;
40 END id_nuevoinstructor;
41 END pac_escuela;
```