

BASES DE DATOS

Guión

Introducción al lenguaje procedimental PL/SQL

Profesor : Héctor Gómez Gauchía

Lenguaje procedimental para consultas PL/SQL

1 Introducción

- Evita necesidad de usar lenguaje externo, c++, java...
- Evita generar muchas peticiones de consultas que llegan al servidor
- Es de Oracle. Muy parecido al estándar PSM (Persistent, Stored Modules)
- Incluye las sentencias procedimentales: control (if, case), iteración(loop), excepciones, y las de SQL
- Un programa es una serie de *bloques*. Un bloque contiene:
 - Declaraciones de variables, cursores,... (DECLARE)
 - Cuerpo (BEGIN): son las sentencias a ejecutar
 - Excepciones: para tratar situaciones previsibles excepcionales.
- Los bloques se pueden anidar. Hay que tener en cuenta la visibilidad de las variables.

Formas de uso:

- **Bloques sin nombre:** Se escriben en el interprete y se ejecutan sin almacenarse.
- **Procedimientos con nombre:** Son bloques almacenados que se pueden ejecutar llamandolos desde otro bloque.
- **Funciones con nombre:** Como procedimientos que devuelven un valor.
- **Disparadores (triggers):** como procedimientos que se activan al detectar el SGBD un evento concreto,p.e.: se inserta una fila en una tabla concreta.

2 Bloques sin nombre

a) Estructura del fichero con varios bloques

```
VAR V_V1
  DECLARE
  ...
  BEGIN
  :V_V1 := valor;
  EXCEPTION
  WHEN...
  WHEN...
  END
/
print V_V1
  DECLARE
  BEGIN
  If :V_V1 = algo then ...
  END
/
print V_V1
```

b) Declaración y uso de: (Lo veremos con ejemplos: *explicadPlus.txt* y del *Urman*)

Variables de acoplamiento o transferencia : VAR nombreVar TIPO; se declaran FUERA del bloque

TIPO puede ser: NUMBER, CHAR(N), VARCHAR2(N), ...CLOB

Variables de sustitución: &nombreV , se asignan antes de enviar al servidor. Las pide el interprete.

Variables de bloque/procedimiento: dnibusca CHAR(8) ;

NombreCL cliente.NombreC%TYPE ;

Instrucciones condicionales: IF-THEN-ELSE-END IF, IF anidados

Instrucciones de iteración/ buques

Cursos para recorrer tablas.

c) Instrucciones de control (if) y de Iteración/Blucles : loop, for, while,..

```
loop
  ----
  if ..... then exit;
  ----
end loop;
```

```
for Contador in [reverse] ValorInicio [.. ValorFinal]
loop
  ---
  ---
end loop;
```

```
TupleVariable cursorX%rowtype; (veremos después)
for TupleVariable in CursorX
loop
  ----
  ----
end loop;
```

```
while condicion....
  -----
end loop;
```

d) Cursores

Cada llamada 'fetch' opera con una tupla de las recuperadas en la consulta definida en 'cursor'

Las variables sobre las que se almacena el fetch pueden ser de fila (row) o de atributo.

Son útiles cuando las operaciones a realizar para cada tupla afectada sean diferentes de acuerdo a algunas condiciones.

declare

Cursor NombreCU is

Select

Sal NombreCU%rowtype;

Begin

open NombreCU;

loop

fetch NombreCU into Sal; [ver ejemplo aparte para las variables de atributo,Prac4-cursor3]

exit when NombreCU%notfound; [alternativa : if NombreCU%notfound exit;]

---- (tratamiento de esa tupla) Sal.NombreAtributo

end loop;

e) Excepciones

Dos tipos: De oracle

Del usuario: se declaran : cliente_listillo EXCEPTION;

Después de activarse termina el bloque.

3 Uso de los procedimientos con nombre

Create procedure NombreProced (**nombreAtrib1** [in| out| inout] **TIPO1** [:= valor],) **as**

NombreVar TipoVar;

begin

[exception

when excep1 then

]

end [NombreProced];

4 Funciones

Create function NombreFuncion (**nombreAtrib1** [in| out| inout] **TIPO1** [:= valor],)

return untipo as

```

    NombreVar TipoVar;
begin
    -----
    ---- return unaExpresión;
[exception
    when excep1 then .....]
end [NombreProced];

```

5 Disparadores: triggers

Como hacer esto?:

EJEMPLO A: Cuando el saldo de una tarjeta en BDejemplo sea menor que -1000 €pasar el cliente a moroso

Bases de datos Activas: reaccionan a eventos y ejecutan automáticamente instrucciones.

Paradigma Evento-Condición-Acción (ECA)

evento: actualiza datos con insert, delete, update

condición (opcional): predicado SQL

acción: secuencia de instrucciones SQL statements o un procedimiento

Mecanismo:

cuando un evento ocurre (triggering)

si la condición se satisface

entonces se ejecuta la acción

Cada trigger se asocia a eventos de UNA SOLA TABLA (on)

→ No se puede consultar o modificar la tabla asociada (o relacionada con foreign key)

en el "on" dentro del trigger (error mutating table)

Formato (en oracle)

```

create trigger NombreTrigger
    Modo Evento {, Evento} of NombreAtributo
    on TablaObjetivo
    [[referencing Referencia] NO USAR
    [ for each row ]
    [ when PredicadoSQL]]
    Bloque entero de PL/SQL

```

Donde:

- *Modo*: before or after
- *Evento*: insert, update, delete
- for each row especifica la granularidad

Granularidad:

- for each row el trigger se activa una vez para CADA fila donde ocurre el evento

Sin esa instrucción: el trigger se activa una vez para TODAS las filas afectadas.

Puede haber problemas de activación al formar ciclos entre varios triggers.

EJEMPLO A:

```

create trigger TR1
    After update OF saldo
    ON tarjeta
    for each row
    when new.saldo < -1000
declare (si hay variables)
begin
    Insert into moroso
values (new.dni, 'empl')
end TR1;

```

Operaciones sobre triggers:

- Ver qué triggers tengo creados: `select trigger_name from user_triggers;`

- Más detalles de un solo trigger:

```
select trigger_type, triggering_event, table_name, trigger_body
from user_triggers
where trigger_name = 'nombreDeTuTrigger'; (poner comillas verticales)
```

- Borrar trigger: `drop trigger nombreDeTuTrigger;`

- Deshabilitar un trigger: `alter trigger nombreDeTuTrigger {disable | enable};`

EJEMPLO B: En la BDejemplo, no permitir que compre nadie con saldo de tarjeta menor de -1000 €. Cuando alguien lo intente se activará una excepción de error.

EJEMPLO C: En la BD de universidad, almacenar en profesores el total de créditos que imparte cada uno y actualizarlos automáticamente cada vez que se modifiquen los créditos de una asignatura.

Asignaturas(codasig, creditos)

Profesores (DNI, Nombre, ToTcred)

Imparte(DNI, codasig)

Ver enunciado prac4/aptdo 3 , dentro, ver código 'trigger3'

REM Oracle8 PL/SQL Programming by Scott Urman.

REM ***** si existe el estudiante actualiza sus datos

REM ***** si no existe, lo crea

DECLARE

/* Declara variables */

v_NewMajor VARCHAR2(10) := 'Bases de Datos';

v_FirstName VARCHAR2(10) := 'Pepito';

v_LastName VARCHAR2(10) := 'Perez';

BEGIN

/* Actualiza tabla de estudiantes */

UPDATE students

SET major = v_NewMajor

WHERE first_name = v_FirstName

AND last_name = v_LastName;

/* si no se encuentra esa fila, la crea insertandola */

/* donde student_sequence.NEXTVAL es una secuencia creada fuera del
bloque */

IF SQL%NOTFOUND THEN

INSERT INTO students (ID, first_name, last_name, major)

VALUES (student_sequence.NEXTVAL, v_FirstName, v_LastName, v_NewMajor);

END IF;

END;

/

Otra Versión:

INSERT INTO students (ID, first_name, last_name, major)

SELECT student_sequence.NEXTVAL, v_FirstName, v_LastName, v_NewMajor

FROM students

WHERE NOT (first_name = v_FirstName

AND last_name = v_LastName);

```
-- ***** fichero: explicadplus *****

-- que enseñe el contenido de este fichero al llamarlo con @
set echo on

-- variable de transferencia para que se vea contenido fuera del
-- bloque
VAR v_DNICL CHAR(8);
VAR v_NombreCL CHAR(30);
VAR v_TelCL CHAR(12);
VAR v_DirCL VARCHAR2(50);

VAR v_coderror NUMBER;
VAR v_texterror VARCHAR2(100);
VAR v_mensa VARCHAR2(50);

VAR v_miseq NUMBER;

DECLARE
-- variable solo visible dentro del bloque

    dnibusca CHAR(8) := '&dniCliente';
    NombreCL   cliente.NombreC%TYPE:= '&nombreCliente';
    TelCL      cliente.Telefono%TYPE := '&TelCliente';
    DirCL      cliente.Direccion%TYPE := '&DirCliente';
    CLiTupla  cliente%rowtype;

    cliente_listillo EXCEPTION;
BEGIN

select MAX(errid) into :v_miseq
    from dicerror;

SELECT DNI, NombreC
    INTO :v_DNICL, :v_NombreCL
    FROM Cliente
    WHERE DNI = dnibusca;

IF :v_DNICL = '00000005' THEN
    RAISE cliente_listillo ;
ELSE
    :v_DNICL := '-- ELSE ';
END IF;

    :v_DNICL := dnibusca;
    :v_NombreCL := NombreCL;
    :v_TelCL := TelCL;
    :v_DirCL := DirCL;
```

```

EXCEPTION
-- WHEN NO_DATA_FOUND THEN
--   :v_coderror:= SQLCODE;
--   :v_texterror:= SUBSTR(SQLERRM,1, 100);

  WHEN cliente_listillo THEN
    :v_coderror:= SQLCODE;
    :v_texterror:= '----- CLIENTE PELIGROSO -----';

  WHEN OTHERS THEN
    :v_coderror:= SQLCODE;
    :v_texterror:= SUBSTR(SQLERRM,1, 100);
    :v_mensa := 'expcepcion OTHERS';

END;
/

print v_coderror
print v_texterror

-- otro bloque , se ejecuta despues del anterior .

DECLARE

BEGIN
  update Cliente
    set   NombreC = :v_NombreCL,
         Telefono = :v_TelCL,
         Direccion = :v_DirCL
  where DNI = :v_DNICL;

  :v_coderror:= SQLCODE;
  :v_texterror:= SUBSTR(SQLERRM,1, 100);
  :v_mensa:= 'Proc 2: actualiza cliente ' || :v_DNICL;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    :v_coderror:= SQLCODE;
    :v_texterror:= SUBSTR(SQLERRM,1, 100);
    :v_mensa := 'Proc2 : sin datos';

  WHEN OTHERS THEN
    :v_coderror:= SQLCODE;
    :v_texterror:= SUBSTR(SQLERRM,1, 100);
    :v_mensa:= 'Proc 2 : otra excepcion';

END;
/

-- FUERA DEL PROC: solo se ven las var de transf ( sin ':' )
-- la instruccion 'print' solo se usa fuera del PROC

print v_coderror
print v_texterror
print v_DNICL
print v_NombreCL
print v_mensa
print v_miseq

```

***** SOLUCIONES *****

EJEMPLO B: En la BD ejemplo, no permitir que compre nadie con saldo de tarjeta menor de -1000 €. Cuando alguien lo intente se activará una excepción de error.

```
create or replace trigger ErrorSaldo
    before insert or update OF saldo ON tarjeta
    for each row
    when (:new.saldo < -1000)
begin
    RAISE_APPLICATION_ERROR(-34867, 'saldo no permitido');
    end if;
end ErrorSaldo;          → Dará un mensaje de error :ORA-34867 saldo no permitido
```

Activación: update tarjeta set saldo = -1001 where NumT = '10000001'

.===== Otra versión del Ejemplo B:

```
create or replace trigger ErrorSaldo
    before insert or update OF saldo ON tarjeta
    for each row
begin
    if (:new.saldo < -1000)
    then RAISE_APPLICATION_ERROR(-34867, 'saldo no permitido');
    end if;
end ErrorSaldo;          → Dará un mensaje de error :ORA-34867 saldo no permitido
```

EJEMPLO C: En la BD de universidad, almacenar en profesores el total de créditos que imparte cada uno y actualizarlos automáticamente cada vez que se modifiquen los créditos de una asignatura.

Se puede hacer mas simple, esto es para mostrar el uso de cursores:

```
create or replace trigger TotCreditos
    After insert or update OF creditos ON asignaturas
    for each row
-- para todas las asignaturas: no hay 'when'
declare
    TDNI CHAR(8);
    CURSOR profesAfectados IS  select DNI
                                from imparte
                                where codasig = :new.codasig;
```

```
begin
    OPEN profesAfectados;
loop
    FETCH profesAfectados into TDNI;
    EXIT WHEN profesAfectados%NOTFOUND;
    update profesores
        set totcred = totcred - :old.creditos + :new.creditos
    where profesores.DNI = TDNI;
end loop;
CLOSE profesAfectados;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.put_line('error imprevisto');
END TotCreditos;
ESTO SE PUEDE HACER SIN CURSOR:

update profesores
    set totcred = totcred - :old.creditos + :new.creditos
where profesores.DNI in
    select DNI
    from imparte
    where codasig = :new.codasig;

end loop;
```

Activación: update asignatura set creditos = 5 where codasig like '6%';