Oracle PL/SQL

Miguel Rodríguez Penabad

Laboratorio de Bases de Datos

Universidade da Coruña

Estructura

Bloques anónimos

```
DECLARE
     /* Sección declarativa - variables PL/SQL, tipos,
        cursores, subprogramas locais */
BEGIN
     /* Sección executable - ordes PL/SQL */
EXCEPTION
     /* Sección de manexo de excepcións  */
END;
```

Exemplo mínimo

```
BEGIN
   NULL;  -- Necesitamos polo menos unha sentencia.
          -- NULL é unha sentencia que non fai nada
END;
/
```

Exemplo 1: Ver a data actual

- SET SERVEROUTPUT ON
- DBMS_OUTPUT.PUT_LINE
- Acabamos o bloque con . e executamos (R ou RUN)
- Se acabamos con /, executa directamente

```
SQL> BEGIN
  2     DBMS_OUTPUT.PUT_LINE(SYSDATE);
  3  END;
  4  .
SQL> R
02/04/06
Procedimiento PL/SQL terminado correctamente.
```


Variables e tipos

Tipos predefinidos

Tipos de SQL máis BOOLEAN, BINARY_INTEGER

```
DECLARE
     Nome VARCHAR2 (20);
     " Unha variable cun nome rarísimo" NUMBER(1);
     Contador BINARY_INTEGER;
     ProcesoFinalizado BOOLEAN;
```

Tipos definidos polo usuario (rexistros)

```
DECLARE
   TYPE t_RexistroEstudiante IS RECORD (
          Nome    VARCHAR2(20),
          Apelido1 VARCHAR2(20),
          Apelido2 VARCHAR2(20)
          );
   variable_Estudiante t_RexistroEstudiante;
```

Variables e tipos (ii)

Sintaxe completa

```
  <nome_variable> [CONSTANT] <tipo>
        [NOT NULL] [{:=|DEFAULT} <valor por defecto>];
```

```
DECLARE
  num1 NUMBER;
  fecha1 DATE :=SYSDATE;
  num3 NUMBER DEFAULT 3;
  num4 CONSTANT NUMBER:=4;
  num5 NUMBER NOT NULL DEFAULT 5;
  num6 CONSTANT NUMBER NOT NULL := 6;
```

Usar o tipo doutra variable, dun atributo ou dunha táboa

```
DECLARE
  salario NUMBER(7,2); -- Problema se cambia a definición na táboa
```

Mellor:

```
DECLARE
  salario EMP.SAL%TYPE;
  soldo salario%TYPE;
  rEmp EMP%ROWTYPE;
```


Sección executable

Manipular variables

- Asignación: :=
- Dependendo do tipo:
  - Operacións matemáticas (+,-,*,/,...), concatenación (||)
  - Funcións (power, upper, length, ...)
  - etc.

```
DECLARE
    n BINARY_INTEGER;
    string VARCHAR(20);
BEGIN
    n := 3;
    DBMS_OUTPUT.PUT_LINE('n vale '||n);
    n := power(2,3);
    DBMS_OUTPUT.PUT_LINE('n vale 2 elvado a 3: '||n);
    string := 'Un tExTo';
    DBMS_OUTPUT.PUT_LINE(string||' en maiúsculas é '|| upper(string));
END;
/
n vale 3
n vale 2 elvado a 3: 8
Un tExTo en maiúsculas é UN TEXTO
```

SQL en bloques PL/SQL

- Podemos incluir variables na sentencia SQL
- Podemos usar DML
  - Insert, delete, update
  - Para o SELECT teremos varios casos
- O uso de DDL será de forma distinta

SQL en bloques PL/SQL (ii)

Exemplo

- Crea unha táboa PERSOA(IDPERSOA,NOME,IDADE)
- Crea un bloque de código para insertar datos
  - Directamente
  - Lendo os datos e almacenándoos nun rexistro

```
create table persoa(idpersoa char(12) not null primary key,
                    nome char(20), idade number(3));

DECLARE
     rPersoa persoa%ROWTYPE;
BEGIN
   INSERT INTO PERSOA(IDPERSOA,NOME,IDADE)
          VALUES ('01234567T', 'Carpanta', 50);
   INSERT INTO PERSOA(IDPERSOA,NOME,IDADE)
          VALUES ('3456789B', 'Sacarino', 17);

   rPersoa.idpersoa   := '12345678A';
   rPersoa.nome  := 'Matusalén';
   rPersoa.idade := 850;
   INSERT INTO PERSOA
          VALUES rPersoa;
END;
```

SQL en bloques PL/SQL (iii)

Exemplo

- Actualiza a idade de todas as persoas, engadindo un ano

```
begin
    UPDATE PERSOA SET IDADE = IDADE + 1;
end;
```

- Borra as persoas cun identificador que acabe en T

```
begin
      DELETE FROM PERSOA WHERE IDPERSOA LIKE '%T';
end;
```


SQL en bloques PL/SQL (iv)

Selección de datos

- Uso da cláusula SELECT ... INTO
- Con variables ou rexistros PL/SQL
- Obtén e imprime información da persoa con IDPERSOA '12345678A'

```
DECLARE
  rPersoa PERSOA%ROWTYPE;
BEGIN
   SELECT *
         INTO rPersoa
         FROM PERSOA
         WHERE IDPERSOA='12345678A';
   DBMS_OUTPUT.PUT_LINE(rPersoa.IDPERSOA ||' '||
                        rPersoa.NOME ||' '||
                        rPersoa.IDADE);
END;
/

12345678A    Matusalén           851
```

SQL en bloques PL/SQL (v)

Selección de datos (cont.)

- Obtén e imprime información da(s) persoa(s) de máis de 15 anos

```
DECLARE
  rPersoa PERSOA%ROWTYPE;
BEGIN
   SELECT * INTO rPersoa
         FROM PERSOA
         WHERE IDADE > 15;
   DBMS_OUTPUT.PUT_LINE(rPersoa.IDPERSOA ||' '||
                        rPersoa.NOME ||' '||
                        rPersoa.IDADE);
END;
```

- Excepción TOO_MANY_ROWS

```
ERROR en línea 1:
ORA-01422: la recuperación exacta devuelve un número
           mayor de filas que el solicitado
ORA-06512: en línea 4
```


SQL en bloques PL/SQL (vi)

Selección de datos (cont.)

- Obtén e imprime información da(s) persoa(s) de 101 anos

```
DECLARE
  rPersoa PERSOA%ROWTYPE;
BEGIN
   SELECT * INTO rPersoa
         FROM PERSOA
         WHERE IDADE = 101;
   DBMS_OUTPUT.PUT_LINE(rPersoa.IDPERSOA ||' '||
                        rPersoa.NOME ||' '||
                        rPersoa.IDADE);
END;
```

- Excepción NO_DATA_FOUND

```
ERROR en línea 1:
ORA-01403: no se han encontrado datos
ORA-06512: en línea 4
```

SQL en bloques PL/SQL (vii)

Selección de datos (cont.)

- O `SELECT ... INTO` só pode usarse se sabemos que imos recuperar unha fila
- Exemplos: funcións de agregación (SUM, AVG, COUNT, ...)

```
DECLARE
  numero NUMBER;
  media  NUMBER;
BEGIN
   SELECT COUNT(*), AVG(IDADE)
        INTO numero, media
         FROM PERSOA;
   DBMS_OUTPUT.PUT_LINE('Hai '|| numero || ' persoas');
   DBMS_OUTPUT.PUT_LINE('A idade media é de '|| numero || ' anos');
END;
/

Hai 3 persoas
A idade media é de 3 anos
```


Estructuras de Control

Condicional

- Simple

```
IF <condición> THEN
   <secuencia que se executa si condición é certa>
END IF;
```

- Sintaxe completa

```
IF <cond1> THEN
   /* Accións */
[ELSIF <cond2> THEN
  /* Accións */
  ...                    ]
[ELSE
  /* Accións se todas as condicións
     anteriores non son certas */ ]
END IF;
```

Estructuras de Control (ii)

Condicionais. Exercicio

- Escribe un bloque de código que indique se a media de idade é superior ou non a 25 anos.
- Funciona `IF (SELECT AVG(IDADE) ....)`?
  - Non
  - Hai que usar `SELECT ... INTO`

```
DECLARE
    media NUMBER;
BEGIN
    SELECT AVG(IDADE) INTO media
        FROM PERSOA;
    IF media > 25 THEN
        DBMS_OUTPUT.PUT_LINE('Idade media superior a 25');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Non é superior a 25');
    END IF;
END;
/
Idade media superior a 25
```


Estructuras de Control (iii)

Bucles

- Simple: `LOOP ... END LOOP`

```
DECLARE
    contador BINARY_INTEGER := 1;
BEGIN
   LOOP -- Bucle infinito!
     DBMS_OUTPUT.PUT_LINE(contador);
     contador := contador + 1;
   END LOOP;
END;
```

Estructuras de Control(iv)

Bucles

- Introducir unha condición e usar EXIT

```
DECLARE
    contador BINARY_INTEGER := 1;
BEGIN
   LOOP
     IF contador > 10 THEN
          EXIT;
     END IF;
     DBMS_OUTPUT.PUT_LINE(contador);
     contador := contador + 1;
   END LOOP;
END;
```

- Mellor: Usar EXIT WHEN <condicion>

```
   LOOP
     EXIT WHEN contador > 10;
     ...
   END LOOP;
```


Estructuras de Control(iv)

Bucles anidados. Etiquetas

Etiqueta: <<nome_etiqueta>> Localiza a seguinte liña de código.

```
DECLARE
      cont1 BINARY_INTEGER := 1;
      cont2 BINARY_INTEGER := 1;
BEGIN
     <<externo>>
     LOOP
       DBMS_OUTPUT.PUT_LINE('Bucle externo: '||cont1);
       cont2:=1;
       <<interno>>
       LOOP
          DBMS_OUTPUT.PUT(' Int:'||cont2); cont2:=cont2+1;
          EXIT interno WHEN cont2>cont1;
       END LOOP;
       DBMS_OUTPUT.NEW_LINE; cont1 := cont1 + 1;
       EXIT externo WHEN cont1 > 10;
    END LOOP;
END;
```

Estructuras de Control(v)

Bucles WHILE

- Sintaxe: `WHILE <condicion> LOOP ... END LOOP;`
- Comproba a condición antes de entrar no bucle
- Exemplo:

```
DECLARE
    contador BINARY_INTEGER := 1;
BEGIN
   WHILE contador <= 10 LOOP
     DBMS_OUTPUT.PUT_LINE(contador);
     contador := contador + 1;
   END LOOP;
END;
```


Estructuras de Control(vi)

Bucles FOR numéricos

- Sintaxe:

```
FOR <variable> IN [REVERSE] <lim_inf> .. <lim_sup> LOOP
      /* Sentencias */
 END LOOP;
```

- A variable do FOR é local e non se declara
- Sempre `<lim_inf> .. <lim_sup>` (incluso con reverse)

```
BEGIN
   FOR contador IN 1..10 LOOP
     DBMS_OUTPUT.PUT_LINE(contador);
   END LOOP;

   -- O seguinte contador é "distinto"
   -- Imprimirá os números do 10 ó 1
   -- ¡for contador in 10..1 loop sería incorrecto!
   FOR contador IN REVERSE 1..10 LOOP
     DBMS_OUTPUT.PUT_LINE(contador);
   END LOOP;
END;
```

Consultas e cursores (i)

Unha fila

- Se sabemos a priori que a consulta devolve exactamente unha fila:

```
SELECT <lista_de_atributos> INTO <lista_de_variables>;
SELECT <lista_de_atributos> INTO <rexistro PL/SQL>;
```

Número indeterminado de filas

- Uso de CURSORes
  1. Declarar cursor
  2. Abrir cursor
  3. (Bucle) Procesar filas
  4. Cerrar cursor

Consultas e cursores (ii)

Declaración

- Na sección de declaración de variables
- Sintaxe: `CURSOR <nome_cursor> IS <consulta>;`
  - Aparece o tipo `<nome_cursor>%ROWTYPE`.
- A consulta pode incluir variables

Exemplos

```
DECLARE
  CURSOR c_xente IS
      SELECT * FROM PERSOA;

  CURSOR c_xubilados IS
      SELECT * FROM PERSOA
            WHERE IDADE >= 65;

  limiteIdade PERSOA.IDADE%TYPE;
  CURSOR c_xente_maior_de IS
      SELECT * FROM PERSOA
            WHERE IDADE >= limiteIdade;
  rPersoa c_xente_maior_de%ROWTYPE; -- Sería igual que PERSOA%ROWTYPE
```

Consultas e cursores (iii)

Apertura do cursor

- Abrir o cursor: `OPEN <nome_cursor>;`
- Se a consulta incluía variables debemos darlles o valor antes de abrir o cursor.
- Oracle executa a consulta e usa a *área e contexto*
  - Información sobre a consulta
  - Os datos obtidos da consulta ("conxunto activo")

Exemplo

```
DECLARE
    limiteIdade PERSOA.IDADE%TYPE;
    CURSOR c_xente_maior_de IS
         SELECT * FROM PERSOA WHERE IDADE >= limiteIdade;
BEGIN
    limiteIdade := 65;
    OPEN c_xente_maior_de;
    ...
END;
```


Consultas e cursores (iv)

Cerre do cursor

- Cerrar o cursor: `CLOSE <nome_cursor>;`
- Oracle libera a área de contexto asociada

Exemplo

```
DECLARE
    limiteIdade PERSOA.IDADE%TYPE;
    CURSOR c_xente_maior_de IS
         SELECT * FROM PERSOA WHERE IDADE >= limiteIdade;
    rPersoa c_xente_maior_de%ROWTYPE;
BEGIN
    limiteIdade := 35;
    OPEN c_xente_maior_de;
    ...
    CLOSE c_xente_maior_de;
END;
```

Consultas e cursores (v)

Procesar filas

- Sentencia para obter unha fila:

```
FETCH <nome_cursor>
  INTO {<lista de variables>|<rexistro PL/SQL>};
```

- Normalmente dentro dun bucle

Propiedades dos cursores

%ISOPEN:
- Indica se o cursor está aberto.
- Se <nomecursor>%ISOPEN é certo, podemos consultar as seguintes propiedades.

%FOUND
- O último FETCH obtivo unha fila.

%NOTFOUND
- O último FETCH non obtivo ningunha fila.

%ROWCOUNT
- Número de filas procesadas actualmente.

- As tres últimas propiedades tenen un significado especial para o cursor SQL

Consultas e cursores (vi)

Procesar filas (bucle estándar)

```
DECLARE
    limiteIdade PERSOA.IDADE%TYPE;
    CURSOR c_xente_maior_de IS
         SELECT * FROM PERSOA WHERE IDADE >= limiteIdade;
    rPersoa c_xente_maior_de%ROWTYPE;
BEGIN
    limiteIdade := 35;
    OPEN c_xente_maior_de;
    LOOP
        FETCH c_xente_maior_de
            INTO rPersoa;
        EXIT WHEN c_xente_maior_de%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(rPersoa.idpersoa||', '||rPersoa.nome||', '||rPersoa.idade);
    END LOOP;
    CLOSE c_xente_maior_de;
END;
 /

01234567T   , Carpanta            , 51
12345678A   , Matusalén           , 851
```

Consultas e cursores (vii)

Procesar filas (bucle while)

Lectura adiantada:

```
DECLARE
    limiteIdade PERSOA.IDADE%TYPE;
    CURSOR c_xente_maior_de IS
         SELECT * FROM PERSOA
            WHERE IDADE >= limiteIdade;
    rPersoa c_xente_maior_de%ROWTYPE;
BEGIN
    limiteIdade := 35;
    OPEN c_xente_maior_de;

    FETCH c_xente_maior_de  INTO rPersoa;
    WHILE c_xente_maior_de%FOUND LOOP
        DBMS_OUTPUT.PUT_LINE(rPersoa.idpersoa||', '||rPersoa.nome||', '||rPersoa.idade);
        FETCH c_xente_maior_de  INTO rPersoa;
    END LOOP;

    CLOSE c_xente_maior_de;
END;
```


Consultas e cursores (viii)

Procesar filas (bucle for con cursor explícito)

- Sintaxe:

```
FOR <rexistro> IN <nome_cursor> LOOP
    /* Procesar fila actual */
END LOOP;
```

- Consideracións importantes
  - O <rexistro> non se declara, é interno ó FOR
  - Non abrimos (OPEN) o cursor antes do bucle
  - Non cerramos (CLOSE) o cursor despois do bucle
  - Non facemos FETCH dentro do bucle
  - Non facemos comprobación (%NOTFOUND) de finalización

```
DECLARE
    limiteIdade PERSOA.IDADE%TYPE;
    CURSOR c_xente_maior_de IS
         SELECT * FROM PERSOA WHERE IDADE >= limiteIdade;
BEGIN
    limiteIdade := 35;
    FOR rPersoa IN c_xente_maior_de LOOP
        DBMS_OUTPUT.PUT_LINE(rPersoa.idpersoa||', '||rPersoa.nome||', '||rPersoa.idade);
    END LOOP;
END;
```

Consultas e cursores (ix)

Procesar filas (bucle for con cursor implícito)

- Sintaxe:

```
FOR <rexistro> IN (<consulta>) LOOP
    /* Procesar fila actual */
END LOOP;
```

- Consideracións
  - Comportamento similar ó bucle `FOR` con cursor explícito.
  - Non declaramos o cursor, usamos directamente a consulta.
  - A consulta **debe** ir entre paréntesis.

```
DECLARE
    limiteIdade PERSOA.IDADE%TYPE;
BEGIN
    limiteIdade := 35;
    FOR rPersoa IN (SELECT * FROM PERSOA WHERE IDADE >= limiteIdade)
    LOOP
        DBMS_OUTPUT.PUT_LINE(rPersoa.idpersoa||', '||rPersoa.nome||', '||rPersoa.idade);
    END LOOP;
END;
```


Consultas e cursores (x)

Cursor implícito "SQL"

- Usado para `INSERT`, `UPDATE`, `DELETE` e `SELECT INTO`
- Non abrimos, cerramos, nin facemos `FETCH`
- Podemos consultar as propiedades

  `SQL%FOUND`: hai filas afectadas
  `SQL%NOTFOUND`: non hai filas afectadas
  `SQL%ROWCOUNT`: nº de filas afectadas
  (`SQL%ISOPEN` sempre é falso)

Exemplo

Borra todas as persoas que teñan menos de 10 anos. Se non hai ningunha, amosa unha mensaxe indicándoo.

```
BEGIN
   DELETE FROM PERSOA WHERE IDADE <10;
   IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('Non hai persoas menores de 10 anos');
   END IF;
END;
```

Consultas e cursores (xi)

Cursores actualizables

- Declaramos o cursor `FOR UPDATE [NOWAIT]`
- Bloquéanse todas as filas recuperadas (ata o commit).
- Especificando `NOWAIT` obtemos excepción se non pode bloquear as filas.
- Usamos `WHERE CURRENT OF <nome_cursor>` como condición para actualizar a fila actual

Exemplo

Restar 1 á Idade das persoas con 77 anos.

```
DECLARE
   CURSOR c_persoas IS
          SELECT * FROM PERSOA
               FOR UPDATE NOWAIT;
BEGIN
   FOR rPersoa in c_persoas LOOP
       IF rPersoa.idade = 77 THEN
         UPDATE PERSOA SET IDADE = IDADE - 1
            WHERE CURRENT OF c_persoas;
       END IF;
   END LOOP;
END;
```


SQL dinámico (i)

Sentencias DDL

- Sintaxe:

```
BEGIN
   EXECUTE IMMEDIATE <cadena de caracteres>;
END;
```

Exemplo

```
DECLARE
   comando char(150);
BEGIN
   comando := 'create table tabla (campo char)';
   EXECUTE IMMEDIATE comando;

   EXECUTE IMMEDIATE 'create index idxcampo on tabla(campo)';
END;
```

SQL dinámico (ii)

Exemplo

Problemas mezclando SQL estático e dinámico:
O bloque PL/SQL non compila se tratamos de usar a táboa que se vai crear con SQL dinámico.

```
DECLARE
     comando char(150);
BEGIN
     comando := 'create table tabla (campo char)';
     EXECUTE IMMEDIATE comando;
     EXECUTE IMMEDIATE 'create index idxcampo on tabla(campo)';
  -- Fallará
     INSERT INTO TABLA VALUES('A');
END;
/
   INSERT INTO TABLA VALUES('A');
               *
ERROR en línea 8:
ORA-06550: línea 8, columna 16:
PL/SQL: ORA-00942: la tabla o vista no existe
ORA-06550: línea 8, columna 4:
PL/SQL: SQL Statement ignored
```


SQL dinámico (iii)

Exercicio

Modifica o código anterior de forma que funcione correctamente.

```
DECLARE
     comando char(150);
BEGIN
     comando := 'create table tabla (campo char)';
     EXECUTE IMMEDIATE comando;
     EXECUTE IMMEDIATE
        'create index idxcampo on tabla(campo)';

     EXECUTE IMMEDIATE
         'INSERT INTO TABLA VALUES(''A'')';
END;
```

SQL dinámico (iv)

Variables tipo cursor

- Tipo xenérico: `SYS_REFCURSOR` ou `REF CURSOR`
  - Poden usarse con SQL dinámico.
- Tipo específico: `REF CURSOR RETURN <tipo>`
  - Uso: paso de coleccións de datos como parámetros en subprogramas.
    (Non sirven para SQL dinámico) (Non os estudiaremos)
- Declaramos cursor dese tipo específico
- Construimos a cadea de caracteres da sentencia
- Abrimos o cursor para a sentencia

```
DECLARE
   -- Definición alternativa:
   --  refc ref cursor;  c_xente refc;
   c_xente SYS_REFCURSOR;
   rex PERSOA%ROWTYPE;
BEGIN
   OPEN c_xente FOR 'SELECT * FROM PERSOA';
   LOOP
       FETCH c_xente INTO rex;
       EXIT WHEN c_xente%NOTFOUND;
       DBMS_OUTPUT.PUT_LINE(rex.idpersoa||', '||rex.nome||', '||rex.idade);
   END LOOP;
   CLOSE c_xente;
END;
```


Excepcións (i)

SQLCODE

Informa sobre a última operación SQL do bloque de código PL/SQL

- `SQLCODE = 0` : última operación con éxito
- `SQLCODE < 0` : Erro na última operación

SQLERRM

Mensaxe de erro correspondente ó SQLCODE

```
BEGIN
     DBMS_OUTPUT.PUT_LINE('Código: '||SQLCODE);
     DBMS_OUTPUT.PUT_LINE('Mensaxe: '||SQLERRM);
END;
/
Código: 0
Mensaxe: ORA-0000: normal, successful completion
```

Excepcións (ii)

Estrutura dun bloque PL/SQL

```
BEGIN
     /* Código problemático */
EXCEPTION
   [ WHEN <nome_excepcion1> THEN
       -- Accións se ocurriu excepcion1 ]
   [ WHEN <nome_excepcion2> THEN
       -- Accións se ocurriu excepcion2 ]
   [ WHEN OTHERS THEN
       -- Accións se ocurriu calquera excepcion ]
END;
```

- Necesitamos polo menos un `WHEN ...`
  - Usamos o nome da excepción (`NO_DATA_FOUND`, `TOO_MANY_ROWS`, `ZERO_DIVIDE`, `DUP_VAL_ON_INDEX`, ...)
  - Usamos `OTHERS` para o caso xenérico
- Compróbanse por orde
- Dentro de cada excepción, escribimos código PL/SQL normal
  - `RAISE` relanza a excepción.

Excepcións (iii)

Exemplo

```
  DECLARE
    NOME PERSOA.NOME%TYPE;
  BEGIN
    SELECT NOME
           INTO NOME
           FROM PERSOA
           WHERE IDADE = 101;
  EXCEPTION
  WHEN NO_DATA_FOUND THEN
       DBMS_OUTPUT.PUT_LINE('Non atopei persoas de 101 anos');
  WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE('Erro:    '||SQLCODE);
       DBMS_OUTPUT.PUT_LINE('Mensaxe: '||SQLERRM);
  END;
/
Non atopei persoas de 101 anos
Procedimiento PL/SQL terminado correctamente.
```

Excepcións (iv)

Exemplo

Crea un bloque de código que trate de insertar unha persoa, de xeito que non dea erro se o identificador (clave primaria) existe (por suposto, non se fai a inserción)

```
BEGIN
    INSERT INTO PERSOA(IDPERSOA, NOME, IDADE)
         VALUES('1','persoa que existe',1);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN NULL;
END;

-- Exercicio: comprobar sen usar DUP_VAL_ON_INDEX
```


Excepcións (v)

Excepcións de usuario

Hai 2 formas:

- Excepcións con nome
  - Declarar variable de tipo `EXCEPTION`
  - Elevar (`RAISE`) a excepción baixo certas condicións
  - Pode ser capturada con `WHEN <nome_excepcion> THEN`
- Excepción sen nome, con SQLCODE e SQLERR
  - `RAISE_APPLICATION_ERROR(<codigo>,<mensaxe>)`
  - O <codigo> debe ser negativo, menor que -20000

Excepcións (vi)

Excepcións de usuario: exemplos

Amosar o número de xubilados. Se hai máis de 100, elevar unha excepción e xestionala indicando que hai demasiados.

```
DECLARE
  demasiados_xubilados EXCEPTION;
  numero NUMBER;
BEGIN
   SELECT COUNT(*)
      INTO numero
      FROM persoa
      WHERE idade>65;
   IF numero > 100 THEN
         RAISE demasiados_xubilados;
   END IF;
   DBMS_OUTPUT.PUT_LINE('Hai '||numero||' xubilados.');
EXCEPTION
    WHEN demasiados_xubilados THEN
      DBMS_OUTPUT.PUT_LINE('Hai demasiados xubilados');
END;
```


Excepcións (vii)

Excepcións de usuario: exemplos

Engade unha persoa. Se a idade é negativa, xenera unha excepción con código -20001 e mensaxe 'Idade inválida'.
Non xestiones esa excepción, para que a execución do bloque falle.

```
DECLARE
  rPersoa PERSOA%ROWTYPE;
BEGIN
  rPersoa.IDPERSOA := '32233223K';
  rPersoa.NOME := 'nome';
  rPersoa.IDADE := -1;

  IF rPersoa.IDADE < 0 THEN
     RAISE_APPLICATION_ERROR(-20001,'Idade inválida');
  END IF;
  INSERT INTO PERSOA VALUES rPersoa;
END;
/

ERROR en línea 1:
ORA-20001: Idade inválida
ORA-06512: en línea 9
```

PL/SQL: Procedementos, funcións e paquetes

Miguel Rodríguez Penabad

Laboratorio de Bases de Datos

Universidade da Coruña

Bloques de código en PL/SQL

- Bloques anónimos
  (xa vistos no punto anterior)
- Rutinas almacenadas
  (SQL/PSM: Persistent Stored Modules en SQL:2003)
  - Procedementos (`PROCEDURE`)
    Non devolve ningún valor
  - Funcións (`FUNCTION`)
    Devolve un valor
  - Paquetes (`PACKAGE`)
    Permite construir bibliotecas de procedementos e funcións
- Información no catálogo de Oracle, nas vistas (entre outras)
  - `USER_OBJECTS (OBJ)`, `ALL_OBJECTS`
  - `USER_PROCEDURES`, `ALL_PROCEDURES`
  - `USER_SOURCE`, `ALL_SOURCE`
  - Orde de SQL*Plus `DESC[RIBE] <modulo>`

Procedementos

Creación

```
CREATE [OR REPLACE]
    PROCEDURE <nome_proc> (<lista de parámetros>)
[AUTHID {DEFINER|CURRENT_USER}]
{IS|AS}
<bloque PL/SQL>
```

- Parámetros:
  `<nome> <especificacion E/S> <tipo> [<valor predeterminado>]`
  - A especificación de entrada/saída pode ser IN (predeterminado), OUT, IN OUT
  - O tipo dos argumentos pode ser
    - Xenérico (p.ex. `CHAR`, non `CHAR(10)`)
    - Derivado dun atributo dunha táboa (p.ex. `PERSOA.IDPERSOA%TYPE`)
- Authid: ¿Con que permisos se executa?
  `DEFINER`: cos permisos do creador (predeterminado)
  `CURRENT_USER`: cos permisos do usuario que o executa
- Bloque PL/SQL:
  - Non inclúe a palabra `DECLARE` antes da declaración de variables
  - Acaba en `END;` ou en `END <nome_proc>;`

Procedementos (ii)

Exemplo

Engadir unha persoa da que sabemos o identificador, nome e data de nacemento.

```
CREATE OR REPLACE
    PROCEDURE InsPersoa(oID IN PERSOA.IDPERSOA%TYPE,
                        oNome IN PERSOA.NOME%TYPE,
                        DataNac DATE)
IS
  aIdade NUMBER(3);
BEGIN
  aIdade := MONTHS_BETWEEN(SYSDATE,DataNac)/12;
  INSERT INTO PERSOA (IDPERSOA,NOME,IDADE)
       VALUES(oID,oNome,aIdade);
END InsPersoa;
/

Procedimiento creado.
```

Procedementos (iii)

Creación, borrado, e uso

- Se Oracle nos di "Creado con errores de compilación"

  `SHOW ERRORS`, ou
  `SHOW ERRORS PROCEDURE <nome_proc>`
- Para borrar o procedemento:

  `DROP PROCEDURE <nome_proc>`
- Para executar directamente o procedemento:
  - Nun bloque PL/SQL, chámase directamente.
  - Desde SQL*Plus ("modo SQL", non PL/SQL)
    - `CALL InsPersoa('5','2','21/12/1900');`
    - `EXEC[UTE] InsPersoa('5','2','21/12/1900');`
      que se expande a

```
BEGIN
    InsPersoa('5','2','21/12/1900');
END;
```


Procedementos (iv)

Valores predeterminados e chamadas

```
CREATE OR REPLACE PROCEDURE pordefecto(v1 int default 3, v2 int default 4)
  IS BEGIN
    dbms_output.put_line('v1: '||v1||', v2: '||v2);
  END;
```

- Os argumentos con valores predeterminados ó final.
- Omitindo un parámetro na chamada, o argumento toma o valor predeterminado.

```
begin
 pordefecto(1,2); pordefecto(1); pordefecto();
end;
/
v1: 1, v2: 2
v1: 1, v2: 4
v1: 3, v2: 4
```

- Novidade Oracle 11g: indicar os nomes dos argumentos na chamada (pre 11: só en packages)
  - Calquera argumento pode ter valor predeterminado
  - Reordenamos/omitimos parámetros

```
begin
  pordefecto(v2 => 77, v1 => 10);  pordefecto(v2 => 77);
end;
/
v1: 10, v2: 77
v1: 3, v2: 77
```

Funcións

Creación

```
CREATE [OR REPLACE]
    FUNCTION <nome_func> (<lista de parámetros>)
        RETURN <tipo>
    [AUTHID {DEFINER|CURRENT_USER}]
{IS|AS}
<bloque PL/SQL>
```

- Debemos indicar o tipo (xenérico) que devolve
- Bloque PL/SQL: Debe incluir `RETURN <valor>`
- Se a creamos con erros de compilación:
  - `SHOW ERRORS`, ou `SHOW ERRORS FUNCTION <nome_func>`
- Para borrar unha función: `DROP FUNCTION <nome_func>`
- Para executar:
  - Non se pode directamente como os procedementos
  - En sentencias `SELECT`

    (Excepto cando a función usa DML que modifica datos)
  - En PL/SQL (`variable := funcion(argumentos)`, ...)

Funcións (ii)

Exemplo

Función que calcula o ano de nacemento (aprox.) a partir da idade.

```
CREATE OR REPLACE FUNCTION AnoNac(idade IN PERSOA.IDADE%TYPE)
   RETURN NUMBER
IS
BEGIN
  RETURN to_number(to_char(sysdate,'yyyy')) - idade;
END AnoNac;
/
Función creada.

SELECT Idade, AnoNac(Idade)
    FROM PERSOA;

     IDADE ANONAC(IDADE)
---------- -------------
        51          1957
       851          1157
        17          1991
```

Funcións (iii)

Uso en vistas

```
CREATE VIEW V_PERSOA
   AS SELECT IDPERSOA, NOME, AnoNac(IDADE) ANO_NAC
        FROM PERSOA;
```

Uso en condicionais

- Función que modifica datos
- Devolve BOOLEAN

  true se foi posible
  false se non o foi

Paquetes

Definición da interfaz

```
CREATE [OR REPLACE] PACKAGE <nome_paquete> {IS|AS}
   -- Declaración de variables globais do paquete
   ...
   -- Prototipos de procedementos e funcións
   PROCEDURE <nome_proc1>(<parámetros);
   FUNCTION <nome_func1> (<parámetros>) RETURN <tipo>;
   ...
END <nome_paquete>;
```

Definición da implementación

```
CREATE [OR REPLACE] PACKAGE BODY <nome_paquete> {IS|AS}
   -- Declaración de variables privadas do paquete
   ...
   -- Implementación de procedementos e funcións
   PROCEDURE <proc_privado>(<parámetros>) IS <bloque PL/SQL>
   ...
   PROCEDURE <nome_proc1>(<parámetros>) IS <bloque PL/SQL>
   FUNCTION <nome_func1>(<parámetros>) RETURN <tipo> IS <bloque PL/SQL>
   ...
END <nome_paquete>;
```

Uso

```
<nome_paquete>.<nome_proc1>(<args>)
```

Exemplo: `DBMS_OUTPUT.PUT_LINE('texto')`

Triggers en Oracle

Miguel Rodríguez Penabad

Laboratorio de Bases de Datos

Universidade da Coruña

Creación de triggers

- Para crear ou modificar:

```
CREATE [OR REPLACE] TRIGGER <nome_trigger>
   {BEFORE|AFTER|INSTEAD OF}
               <Evento> [OR <Evento>...] ON <Tabla>
   [ REFERENCING [NEW AS tupla_nova]
                  [OLD AS tupla]_vella] ]
   [FOR EACH ROW]
   [WHEN (<Condicion_SQL_simple>)]
{ <Bloque PL/SQL> | CALL <procedemento(argumentos)> }
```

- Para borrar:

```
DROP TRIGGER <nome_trigger>
```

- Para activar ou desactivar:
  - Triggers individuais

    `ALTER TRIGGER nome_trigger> {ENABLE|DISABLE}`

  - Todos os triggers asociados a unha táboa

    `ALTER TABLE <táboa> {ENABLE|DISABLE} ALL TRIGGERS`

Creación de triggers

- Para consultar os existentes:
  Táboas `ALL_TRIGGERS`, `USER_TRIGGERS`

```
SQL> desc user_triggers
 Nombre                        ¿Nulo?   Tipo
 ----------------------------- -------- -------------
 TRIGGER_NAME                           VARCHAR2(30)
 TRIGGER_TYPE                           VARCHAR2(16)
 TRIGGERING_EVENT                       VARCHAR2(227)
 TABLE_OWNER                            VARCHAR2(30)
 BASE_OBJECT_TYPE                       VARCHAR2(16)
 TABLE_NAME                             VARCHAR2(30)
 COLUMN_NAME                            VARCHAR2(4000)
 REFERENCING_NAMES                      VARCHAR2(128)
 WHEN_CLAUSE                            VARCHAR2(4000)
 STATUS                                 VARCHAR2(8)
 DESCRIPTION                            VARCHAR2(4000)
 ACTION_TYPE                            VARCHAR2(11)
 TRIGGER_BODY                           LONG
```

```
select trigger_name, trigger_type, triggering_event, when_clause, status
   from user_triggers
   where table_name='EMP2';
TRIGGER_NAME    TRIGGER_TYPE    TRIGGER WHEN_CLAUSE     STATUS
--------------- --------------- ------- --------------- --------
EMP_SAL_POS     BEFORE EACH ROW UPDATE  nova.SAL<0      ENABLED

1 fila seleccionada.
```


Comparación co estándar SQL:2003

Evento

- Tipos de evento
  - Eventos DML (`INSERT`, `UPDATE`, `DELETE`) (como SQL:2003)
  - Eventos DDL (`CREATE ON SCHEMA`, ...)
  - Eventos de base de datos (`LOGON`, `SERVERERROR`, ...)
- Número de eventos que controla un trigger
  - Oracle permite controlar máis dun evento (usando `OR`)
  - Co mesmo tempo de activación e a mesma granularidade
  - Podemos saber o evento que o disparou:
    Usamos os predicados `INSERTING`, `UPDATING`, `DELETING`
- Momento de activación
  - `BEFORE`, `AFTER` (como SQL:2003)
  - `INSTEAD OF` (só para vistas)
- Táboas e variables de transición:
  - Non usa táboas de transición (`old/new table`)
  - Si que usa variables de transición (`old/new row`)
    - Nomes por defecto: `new/old`
    - Poden usarse na condición e na acción do trigger.

Comparación co estándar SQL:2003 (cont.)

Condición

- Podemos usar as variables de transición `NEW` e `OLD`
- Non admite sentencias `SELECT`
- Solución:
  - Omitir condición (omitir cláusula `WHEN`)
  - Incluir a condición nun `IF` na acción

Acción

- Bloque PL/SQL (ou chamada a un procedemento).
- Non se permiten sentencias de control transaccional (`COMMIT`, `ROLLBACK`, ...)
- As variables de transición son consideradas *bind variables*

  Irán precedidas do símbolo dous puntos
  (`:NEW` e `:OLD` se non usamos nomes alternativos)
- Definidas para todos os tipos de trigger a nivel de fila
  - `NEW` contén o valor antigo, e `OLD` o novo
    - En triggers de inserción, `OLD` será `NULL`
    - En triggers de borrado, `NEW` será `NULL`
  - Coñécese o valor nos triggers `BEFORE` e `AFTER`
    - `OLD` nunca pode ser modificado
    - `NEW` pode modificarse nos triggers `BEFORE`

Exemplos

Táboas de referencia

```
CREATE TABLE PRODUTO(
  CODPROD NUMERIC(3) NOT NULL,
  NOMPROD   VARCHAR(40),
  PREZO  NUMERIC(6,2),
  CONSTRAINT PK_PRODUTO PRIMARY KEY(CODIGO) );

CREATE TABLE FACTURA(
  NUMERO CHAR(5),
  DATA DATE DEFAULT SYSDATE,
  TOTAL NUMERIC(7,2),
  CONSTRAINT PK_FACTURA PRIMARY KEY(NUMERO) );

CREATE TABLE LINA(
  NUMFAC CHAR(5),
  NUMLI  NUMERIC(3),
  CODPROD NUMERIC(3),
  CANTIDADE NUMERIC(2),
  PREZO NUMERIC(6,2),
  SUBTOTAL NUMERIC(7,2),
  CONSTRAINT PK_LINA PRIMARY KEY(NUMFAC,NUMLI),
  CONSTRAINT FK_FAC FOREIGN KEY(NUMFAC)
      REFERENCES FACTURA(NUMERO) ON DELETE CASCADE,
  CONSTRAINT FK_PROD FOREIGN KEY(CODPROD)
      REFERENCES PRODUTO(CODPROD) ON DELETE CASCADE );
```

Exemplos

Actualización de prezos

Non se permite que unha actualización asigne un prezo negativo a un produto.

```
CREATE OR REPLACE TRIGGER prod_prezo_positivo
    BEFORE UPDATE ON PRODUTO
    REFERENCING NEW AS tupla_nova OLD AS tupla_vella
    FOR EACH ROW
    WHEN (tupla_nova.Prezo < 0)
BEGIN
    :tupla_nova.Prezo := :tupla_vella.Prezo;
END;
```

```
SQL> SELECT * FROM PRODUTO;
    CODIGO NOME                      PREZO
---------- -------------------- ----------
         1 TORNILLO                    ,23

SQL> UPDATE PRODUTO SET PREZO=-2;
1 fila actualizada.

SQL> SELECT * FROM PRODUTO;
    CODIGO NOME                      PREZO
---------- -------------------- ----------
         1 TORNILLO                    ,23
```


Exemplos

Actualización de prezos + inserción

Non se permite que unha actualización asigne un prezo negativo a un produto. Tampouco permitimos a inserción de prezos negativos. Se se intenta, o prezo queda en 0.

```
CREATE OR REPLACE TRIGGER prod_prezo_positivo
    BEFORE UPDATE OR INSERT ON PRODUTO
    REFERENCING NEW AS tupla_nova OLD AS tupla_vella
    FOR EACH ROW
    WHEN (tupla_nova.Prezo < 0)
BEGIN
   IF UPDATING
        THEN :tupla_nova.Prezo := :tupla_vella.Prezo;
        ELSE :tupla_nova.Prezo := 0;
   END IF;
END;
```

```
SQL> INSERT INTO PRODUTO VALUES(2, 'TUERCA', -2);
1 fila creada.
SQL> select * from produto;
    CODIGO NOME                      PREZO
---------- -------------------- ----------
         1 TORNILLO                    ,23
         2 TUERCA                       0
2 filas seleccionadas.
```

Exemplos

Problemas: táboas mutantes

Actualización de prezos (evitar inserción)

Non se permite unha inserción de produtos con prezos negativos.
Se se intenta, a fila non se insertará.

```
CREATE OR REPLACE TRIGGER prod_prezo_positivo_falla
    AFTER INSERT ON PRODUTO
    FOR EACH ROW
    WHEN (new.Prezo < 0)
BEGIN
   DELETE FROM PRODUTO WHERE codprod = :new.codprod;
END;
```

O trigger créase correctamente, pero:

```
SQL> insert into produto values(4,'Erro', -10);
insert into produto values(4,'Erro', -10)
            *
ERROR en línea 1:
ORA-04091: la tabla MIGUEL.PRODUTO está mutando, puede que el
disparador/la función no puedan verla
ORA-06512: en "MIGUEL.PROD_PREZO_POSITIVO_FALLA", línea 1
ORA-04088: error durante la ejecución del disparador
'MIGUEL.PROD_PEZO_POSITIVO_FALLA'
```


O problema das táboas mutantes

Unha táboa está mutando

- Está modificándose actualmente por unha sentencia DML
  - Triggers a nivel fila (`FOR EACH ROW`)
  - Excepto `BEFORE INSERT` en inserciónns de 1 fila
- Está léndose ou modificándos para garantir integridade referencial.
- Non podemos acceder (`INSERT`, `DELETE`, `UPDATE`) á táboa mutante.

Exemplos de triggers que fallan

- `prod_prezo_positivo_falla` (transparencia anterior): `PRODUTO` está sendo modificado
- `trigger_falla1`: `LINA` está mutando (por `FK_PRODUTO`)
- `trigger_falla2`: `PRODUTO` está mutando (por `FK_PRODUTO`)

```
CREATE TRIGGER trigger_falla1
   AFTER DELETE ON PRODUTO
FOR EACH ROW
DECLARE N NUMBER;
BEGIN
  SELECT COUNT(*) INTO N
     FROM LINA;
END;
-- O seguinte DELETE falla
delete from produto where codigo=1;
```

```
CREATE TRIGGER trigger_falla2
AFTER DELETE ON LINA
FOR EACH ROW
DECLARE N NUMBER;
BEGIN
  SELECT COUNT(*) INTO N
     FROM PRODUTO;
END;
-- O seguinte DELETE falla
delete from produto where codigo=1;
```

O problema das táboas mutantes

Posibles solucións

1. Cambiar a acción (se é posbile)
2. Convertir en trigger a nivel de sentencia
3. Simular táboas de transición
   (combinación de varios triggers e procedementos en packages)

Opción 1: Cambiar acción

```
CREATE OR REPLACE TRIGGER prod_prezo_positivo2
    AFTER INSERT ON PRODUTO
    FOR EACH ROW
    WHEN (new.Prezo < 0)
BEGIN
   RAISE_APPLICATION_ERROR(-20002, 'Prezo inválido');
END;
```

```
SQL> insert into produto values(22,'non se inserta', -2);
insert into produto values(22,'non se inserta', -2)
            *
ERROR en línea 1:
ORA-20002: Prezo inválido
ORA-06512: en "MIGUEL.PROD_PREZO_POSITIVO2", línea 1
ORA-04088: error durante la ejecución del disparador
'MIGUEL.PROD_PEZO_POSITIVO2'
```


O problema das táboas mutantes

Posibles solucións (cont.)

Opción 2: Convertir a trigger de sentencia

- A condición pode variar
- O trigger pode non ser totalmente equivalente
  ¿Que pasa se había prezos negativos anteriores?

```
CREATE OR REPLACE TRIGGER prod_prezo_positivo3
AFTER INSERT ON PRODUTO
BEGIN
    DELETE FROM PRODUTO
        WHERE Prezo<0;
END;
```

```
SQL> select count(*) from produto;
  COUNT(*)
----------
         2
SQL> INSERT INTO PRODUTO VALUES(3,'NON SE INSERTA',-2);
1 fila creada.
SQL> select count(*) from produto;
  COUNT(*)
----------
         2
```

O problema das táboas mutantes

Posibles solucións (cont.)

Opción 3: Simular táboas de transición

- Trigger de fila (`FOR EACH ROW`)
  - `BEFORE <evento>` ou `AFTER <evento>`
  - Almacena as filas nunha táboa temporal
- Trigger de sentencia
  - `AFTER <evento>`
  - Recorre a táboa temporal procesando as filas
  - Remata borrando a táboa temporal
- Uso de package
  - Para declarar a táboa temporal
  - Procedementos almacenar e recorrer as filas

1. Crear paquete

```
CREATE OR REPLACE PACKAGE EVITAR_MUTANTES
AS
 PROCEDURE proc_ins_row(cod IN PRODUTO.Codprod%TYPE,
                        pre IN PRODUTO.Prezo%TYPE);
 PROCEDURE proc_after_insert;
END EVITAR_MUTANTES;
```


O problema das táboas mutantes

Posibles solucións (cont.)

Opción 3: Simular táboas de transición (cont.)

2. Crear corpo do paquete

```
CREATE OR REPLACE PACKAGE BODY EVITAR_MUTANTES AS

 TYPE rex_prod IS RECORD(
        codigo PRODUTO.Codprod%TYPE,
        prezo PRODUTO.Prezo%TYPE );
 TYPE tab_prod_t IS TABLE OF rex_prod;
 tab_prod tab_prod_t := tab_prod_t();

PROCEDURE proc_ins_row(cod IN PRODUTO.Codprod%TYPE,
                   pre IN PRODUTO.Prezo%TYPE) IS BEGIN
    tab_prod.extend;
    tab_prod(tab_prod.last).codprod := cod;
    tab_prod(tab_prod.last).prezo := pre;
END proc_ins_row;

PROCEDURE proc_after_insert IS BEGIN
    FOR i in tab_prod.first .. tab_prod.last LOOP
        DELETE FROM PRODUTO WHERE CODPROD = tab_prod(i).codprod;
    END LOOP;
    tab_prod.delete;
END;
END EVITAR_MUTANTES;
```

O problema das táboas mutantes

Posibles solucións (cont.)

Opción 3: Simular táboas de transición (cont.)

3. Crear os triggers

```
CREATE OR REPLACE TRIGGER prezopos_fila
BEFORE INSERT ON PRODUTO
FOR EACH ROW
WHEN (new.Prezo < 0)
CALL EVITAR_MUTANTES.PROC_INS_ROW(:new.codprod,:new.Prezo)
/
CREATE OR REPLACE TRIGGER prezopos_sentencia
AFTER INSERT ON PRODUTO
CALL EVITAR_MUTANTES.PROC_AFTER_INSERT
/
```

```
SQL> ALTER TABLE PRODUTO DISABLE ALL TRIGGERS;
SQL> INSERT INTO PRODUTO VALUES(3,'INSERTADO',-2);
1 fila creada.

SQL> SELECT * FROM PRODUTO;
   CODPROD NOMPROD                  PREZO
---------- -------------------- ----------
         3 INSERTADO                    -2
SQL> ALTER TABLE PRODUTO ENABLE ALL TRIGGERS;
SQL> INSERT INTO PRODUTO VALUES(4, 'NON INSERTADO', -4);
1 fila creada.
SQL> SELECT * FROM PRODUTO;
   CODPROD NOMPROD                  PREZO
---------- -------------------- ----------
         3 INSERTADO                    -2
```


Triggers non estándar

Triggers `INSTEAD OF`

- Non forman parte do estándar SQL:2003
- Só poden definirse sobre vistas
  - Usados para poder actualizar vistas non actualizables.
  - Sempre son a nivel de fila.

Exemplo

```
CREATE TABLE EMP2
   AS  SELECT * FROM EMP;

CREATE VIEW EMPS_POR_DEP
   AS SELECT D.DEPTNO, DNAME, LOC, COUNT(*) EMPS
         FROM EMP2 E JOIN DEPT D ON E.DEPTNO=D.DEPTNO
         GROUP BY D.DEPTNO, DNAME, LOC;

SQL> SELECT * FROM EMPS_POR_DEP;

DEPTNO DNAME          LOC                 EMPS
------ -------------- ------------- ----------
    20 RESEARCH       DALLAS                 5
    10 ACCOUNTING     NEW YORK               3
    30 SALES          CHICAGO                6
```

Triggers non estándar

Triggers INSTEAD OF (cont.)

Exemplo (cont.)

- A vista non é actualizable.

```
SQL> DELETE FROM EMPS_POR_DEP WHERE DNAME='SALES';
DELETE FROM EMPS_POR_DEP WHERE DNAME='SALES'
            *
ERROR en línea 1:
ORA-01732: operación de manipulación de datos no válida en esta vista
```

- Podemos definir a "regra de actualización" co trigger.

```
CREATE OR REPLACE TRIGGER BORRAR_EN_EMPS_POR_DEP
  INSTEAD OF DELETE ON EMPS_POR_DEP
  FOR EACH ROW
  BEGIN
      DELETE FROM EMP2
          WHERE DEPTNO = :OLD.DEPTNO;
  END;
```

```
SQL> DELETE FROM EMPS_POR_DEP WHERE DNAME='SALES';
1 fila suprimida.
```


Triggers non estándar

Eventos DDL e de base de datos

- Eventos que non controlan modificación de datos
- Usados en auditoría de base de datos

```
CREATE TABLE LOGCREACIONS(
    USUARIO CHAR(20), DATA DATE,
    TIPO_OBX CHAR(20), OBXECTO CHAR(20));

CREATE OR REPLACE TRIGGER LogCreacions
  AFTER CREATE ON SCHEMA
BEGIN
  INSERT INTO LogCreacions(usuario,data,tipo_obx,obxecto)
    VALUES(USER,SYSDATE,ORA_DICT_OBJ_TYPE,ORA_DICT_OBJ_NAME);
END LogCreacions;
/

CREATE TABLE CONEXIONS(
       USUARIO CHAR(20), DATA_HORA DATE, TIPO CHAR(12));

CREATE TRIGGER LOG_CONEXION
      AFTER LOGON ON DATABASE
      BEGIN
      INSERT INTO CONEXIONS(USUARIO,DATA_HORA,TIPO)
         VALUES (USER, SYSDATE, 'CONEXIÓN');
      END LOG_CONEXION;
/
```

Novidades Oracle 11g

Ordenación da execución de triggers

- Útil se existen varios triggers para o mesmo evento/granularidade/tempo activación
  - Cláusula `FOLLOWS`

```
 CREATE OR REPLACE TRIGGER trigger_1
   AFTER UPDATE ON t
   FOR EACH ROW
   BEGIN
     ...
   END trigger_1;

-- O trigger_2 executarase despois do trigger_1,
-- despois de actualizar cada fila de t
 CREATE OR REPLACE TRIGGER trigger_2
   AFTER UPDATE ON t
   FOR EACH ROW
   FOLLOWS trigger_1
   BEGIN
     ...
   END trigger_2;
```


Novidades Oracle 11g

Triggers compostos (compound triggers)

- Permite establecer varias accións
  - Con distinta granularidade
  - Con distinto tempo de activación

```
 CREATE OR REPLACE TRIGGER trigger_comp
      FOR <evento> ON <táboa>
        COMPOUND TRIGGER
<Declaracións (opcionais)>
-- Non é necesario incluir os seguintes catro bloques
     BEFORE STATEMENT IS
     BEGIN   <acción>  END BEFORE STATEMENT;

     BEFORE EACH ROW IS
     BEGIN  <acción>   END BEFORE EACH ROW;

     AFTER EACH ROW IS
     BEGIN  <acción>   END AFTER EACH ROW;

     AFTER STATEMENT IS
     BEGIN  <acción>   END AFTER STATEMENT;
END trigger_comp;
```

Novidades Oracle 11g

Triggers compostos (compound triggers) - exemplo

```
-- Similar ó exemplo que usa un package
CREATE OR REPLACE TRIGGER trigger_comp
     FOR INSERT ON PRODUTO
       COMPOUND TRIGGER
TYPE tab_prod_t IS TABLE OF PRODUTO.Codprod%TYPE;
tab_prod tab_prod_t := tab_prod_t();

    BEFORE EACH ROW IS
    BEGIN
       IF :new.prezo<0  THEN
         tab_prod.extend;
         tab_prod(tab_prod.last) := :new.codprod;
       END IF;
    END BEFORE EACH ROW;

    AFTER STATEMENT IS
    BEGIN
       FOR i in tab_prod.first .. tab_prod.last LOOP
           DELETE FROM PRODUTO WHERE codprod = tab_prod(i);
       END LOOP;
    END AFTER STATEMENT;
END trigger_comp;
```