Sesión 7: PL/SQL:
Conceptos básicos de bloque PL/SQL y cursores. Manejo de excepciones.

CONCEPTOS BÁSICOS DE PL/SQL

BLOQUES PL/SQL → Bloques de sentencias ejecutables por la BD.

Un bloque SQL tiene tres partes: una declarativa, una ejecutable y una última para el manejo de excepciones (warnings y condiciones de error). De estas tres partes, sólo la parte ejecutable es obligatoria.

```
[ DECLARE
        -- declaraciones]
BEGIN
        --  sentencias
[EXCEPTION
        --  manejo de excepciones]
END;
```

DECLARACIÓN de VARIABLES y CONSTANTES

En la parte declarativa se pueden declarar variables y constantes, por ejemplo:

```
DECLARE
total number(6,2);
nombre varchar(30):= 'JUANA';  --  valor por defecto pero se puede modificar
maximo CONSTANT number(4):=9999;    -- No se puede modificar
```

En la parte declarativa, las variables se pueden inicializar a un valor específico o no hacerlo, mientras que las constantes han de ser inicializadas en la parte declarativa.

DECLARACIÓN de VARIABLES y CONSTANTES

***Si necesitamos que el tipo de datos coincida con el definido para una columna de una tabla podemos utilizar* %TYPE**

El atributo %TYPE proporciona el tipo de dato de una variable o columna de la base de datos.

```
DECLARE
    valor number(4,2);
    auxvalor valor%TYPE;
    auxcategoria habitacion.categoria%TYPE;
```

La variable auxvalor es del mismo tipo de datos que valor y
la variable auxcategoria tiene el mismo tipo de datos que la columna categoría de la tabla habitación.

Utilizar el atributo %TYPE hace que no sea necesario conocer el tipo de dato exacto de una variable o columna, y por otro lado, tiene la ventaja de que si la definición del tipo de datos de una columna cambia, el tipo de dato de la columna también cambia automáticamente.

***Si necesitamos definir una estructura similar a una fila de una tabla podemos utilizar* %ROWTYPE**

El atributo %ROWTYPE obtiene un tipo de registro que representa una fila de una tabla. Los campos del registro y las correspondientes columnas de la tabla tienen el mismo nombre y el mismo tipo de datos

```
DECLARE
    pvp  pvptemporada%ROWTYPE;
```

ASIGNAR VALOR A UNA VARIABLE

Por defecto las variables se inicializan a NULL.
Se les puede asignar valor de dos formas: a través de una **expresión**, o a través de una **sentencia SELECT**. ¡**EN ESTE ÚLTIMO CASO LA SELECT SÓLO PUEDE DEVOLVER UNA FILA! En caso contrario se produciría una excepción**

```
aux_valor1:=25;                SELECT categoria  INTO auxcategoria  FROM habitacion WHERE ….
aux_valor2:=total-10;          SELECT cod, descripción  INTO auxcod, auxdesc  FROM ACTIVIDADES WHERE …
```

Para mostrar mensajes podemos utilizar:

MENSAJES en bloques PL/SQL

dbms_output.put_line(mensaje)

Deposita el mensaje en un buffer y **prosigue** con la ejecución

Para que los mensajes se muestren con DBMS_OUTPUT es necesario ejecutar en la sesión de trabajo

SET SERVEROUTPUT ON

Si lo que se quiere es mostrar un mensaje de error y **detener** la ejecución

raise_application_error (número_error, mensaje)
número_error debe ir entre –20000…-20999, el resto está reservado para errores propios de oracle

raise_application_error(-20101,'Alcanzado total de votantes')

ESTRUCTURAS DE CONTROL en bloques PL/SQL

Para trabajar dentro del bloque BEGIN-END se pueden utilizar **estructuras de control** similares a las de otros lenguajes de programación.

ESTRUCTURAS DE CONTROL CONDICIONAL

```
IF … THEN…
END IF;
```

```
IF … THEN…
ELSE …
END IF;
```

```
IF … THEN…
ELSIF … THEN …
ELSIF … THEN …
[ELSE …]
END IF;
```

ESTRUCTURAS DE CONTROL ITERATIVO

```
FOR i IN min .. max LOOP
  …
  …
END LOOP;
```

```
WHILE condición LOOP
  …
END LOOP;
```

```
LOOP
 …
…
…
EXIT WHEN …;
END LOOP;
```

```
LOOP
 …
  IF condición THEN
        …
        EXIT;
  END IF;
…
END LOOP;
```

CURSORES EN PL/SQL

PERMITEN RECORRER UNA SENTENCIA SQL, RECUPERAR INFORMACIÓN FILA A FILA, Y PARA CADA UNA DE ELLAS REALZAR OPERACIONES CON PL/SQL
Cuando se declara un cursor, se le da un nombre y se le asigna a una consulta específica.

CURSOR nombre_cursor IS sentencia_select;

Ejemplo:

```
DECLARE
CURSOR c1 IS SELECT num, pSA FROM habitación h, pvptemporada p  WHERE h.categoria=p.categoria;
```

Ahora es como si el resultado de ejecutar la sentencia SELECT fuera una tabla llamada c1 que podemos ir recorriendo fila a fila.

Trabajando con OPEN, FETCH y CLOSE

Para trabajar con cursores se pueden utilizar los comandos **OPEN, FETCH y CLOSE**.

```
DECLARE
        auxnum      habitación.num%type;
        auxsuperf   habitación.superf%type;
        auxsupMin   categoría.supMin%type;
    CURSOR c1 IS SELECT num, superf, supMin FROM habitación , categoría   WHERE categoria=nombre;
BEGIN
    OPEN c1;
    FETCH c1 INTO auxnum, auxsuperf, auxsupMin;
    Ahora se recorre cursos fila a fila con un bucle y se hace un FETCH
    en cada iteración…
    CLOSE c1;
END;
```

En el bloque DECLARE junto con la definición de variables y constantes, se hace la declaración de los cursores

OPEN
Abre el cursor, es decir, ejecuta la consulta e identifica el resultado (las filas resultantes de la consulta). Al ejecutar OPEN, las filas no se devuelven.

FETCH
Permite devolver las filas una a una.
Cada vez que se ejecuta FETCH el cursor avanza a la siguiente fila del resultado.

Para cada columna que se devuelve en la consulta asociada al cursor, tendremos que tener una variable después del INTO, con un tipo de datos compatible.

CLOSE
Cierra el cursor. Una vez cerrado podría volver a abrirse.

Cada cursor tiene asociados cuatro atributos**: %FOUND, %ISOPEN, %NOTFOUND, %ROWCOUNT**

Tras abrir un cursor con OPEN y antes de que se haya hecho el primer FETCH

- el atributo %FOUND contiene NULL
- el valor de %ROWCOUNT es 0

A partir de ese momento cada vez que hacemos FETCH,

- %FOUND devolverá TRUE si el último FETCH devolvió una fila, o FALSE en caso contrario
- %ROWCOUNT contendrá el número de filas a las que ya se ha realizado FETCH.

El atributo opuesto a %FOUND es %NOTFOUND.

El atributo %ISOPEN devuelve

- true si el cursor está abierto
- false si no está abierto

Ejemplo de uso de estos atributos

```
OPEN c1;
FETCH c1 INTO variable1, variable2;
WHILE  C1%FOUND LOOP
      IF c1%ROWCONT >10  and variable1='PEPE' THEN
           INSERT INTO TABLA_X (campo1, campo2) VALUES (variable1, variable2);
      END IF;
     FETCH c1 INTO variable1, variable2;   --- ¡¡¡ OJO !!! Si no se hace este fetch se entra en un bucle infinito
 END LOOP;
 CLOSE c1;
```

CURSOR EN BUCLE FOR

Al trabajar con cursores, se puede simplificar el código utilizando un bucle FOR **en lugar de** OPEN, FETCH y CLOSE. El bucle FOR:

- **abre implícitamente el cursor**
- **realiza FETCH repetidamente**
- **y cierra el cursor cuando todas las filas han sido procesadas.**

En el bucle además de abrir el cursor se declara una variable (*en el ejemplo que sigue* regc1).

- Esta variable sólo puede ser utilizada dentro del bucle.
- Es una variable de tipo registro, cuyos campos tienen el mismo nombre y tipo de datos que las columnas de la sentencia SELECT que figura en la definición del cursor.
- Si alguna de las columnas fuese calculada sería necesario que tuviese un alias en la sentencia SELECT

En cualquier momento se puede salir del bucle con la sentencia **EXIT**
Ejemplo:

```
DECLARE
    •
    CURSOR c1 IS SELECT num, superf, supMin FROM habitación, categoría   WHERE categoria=nombre;
BEGIN

    FOR regc1 IN c1 LOOP
        IF regc1.superf < regc1.supMin THEN
            Dbms_output.put_line(' Habitación '|| regc1.num ||
             ' dimensión INCORRECTA ***');
            INSERT INTO MALCATEGORIA
                VALUES(regc1.num, regc1.supMin-regc1.superf);
        ELSE
            Dbms_output.put_line(' Habitación '|| regc1.num ||
             ' dimensión correcta');
        END IF;
    END LOOP;
    •
    •
END;
```

En la primera iteración:
- se abre (OPEN) el cursor,
- se accede a la primera fila (FETCH) y
- se deposita en regc1 (queda declarada en el FOR automáticamente)

Cada nueva iteración significa el acceso a la siguiente fila (FETCH) y depositarla en regc1.

Al terminar de recorrer todas las filas, finalizan las iteraciones y se cierra automáticamente el cursor (CLOSE)

CONTROL DE EXCEPCIONES EN BLOQUES EN PL/SQL

En la mayoría de los lenguajes de programación se emplean EXCEPCIONES para el manejo de errores en tiempo de ejecución. En Oracle también podemos definir excepciones. Estas definiciones hacen que, cuando se produce un error, salte una excepción y pase el control a la sección EXCEPTION correspondiente. Las acciones a seguir se incluyen en la sección EXCEPTION definida al final del bloque BEGIN-END.

```
BEGIN

  ...

EXCEPTION
  WHEN <nombre_excepción> THEN
    <bloque de sentencias>;
 WHEN <nombre_excepción> THEN
    <bloque de sentencias>;

  ...
  [WHEN OTHERS THEN <bloque de sentencias>;]
END;
```

Existen dos tipos de excepciones:

- Las excepciones predefinidas por ORACLE.
- Las excepciones definidas por el usuario.

EXCEPCIONES PREDEFINIDAS POR ORACLE

Son aquellas que se disparan automáticamente al producirse determinados errores. Las más frecuentes son (hay muchas más):

- **too_many_rows**: Se produce cuando SELECT … INTO devuelve más de una fila.
- **no_data_found**: se produce cuando un SELECT… INTO no devuelve ninguna fila.
- **value_error**: se produce cuando hay un error aritmético o de conversión.
- **zero_divide**: se puede cuando hay una división entre 0.
- **Dup_val_on_index**: se crea cuando se intenta almacenar un valor que crearía duplicados en la clave primaria o en una columna con restricción UNIQUE.
- **invalid_number**: se produce cuando se intenta convertir una cadena a un valor numérico.

EXCEPCIONES PREDEFINIDAS POR EL USUARIO

Son las que podemos definir nosotros en nuestros módulos PL/SQL. Para trabajar con excepciones definidas por el usuario, es necesario que se realicen 3 pasos:

1. Definición de la etiqueta (o nombre) de la excepción en la sección DECLARE. La sintaxis es:

   Nombre_de_excepción EXCEPTION

2. Lanzar la excepción. Se hace mediante la orden RAISE.
3. Definir las sentencias a ejecutar, cuando se lance la excepción. Esto se hace en la sección EXCEPTION.

   WHEN nombre_de_excepción THEN …

Ejemplos de manejo de excepciones

Ejemplo 1:

Crea un bloque PL/SQL en la que dado el nif 99999999S de un empleado se muestre su nombre y teléfono. Si ese empleado no existe, debe mandarse un mensaje diciendo que no se encuentra en la base de datos, pero continúe la ejecución.

```
DECLARE
        auxnombre empleado.nombre%type;
        auxtelefono empleado.telefono%type;
BEGIN
  select nombre, telefono into auxnombre, auxtelefono
  from empleado where nif='99999999S';
  dbms_output.put_line('Nombre: '||auxnombre||'   teléfono: '||auxtelefono);
  EXCEPTION
     WHEN no_data_found then
      dbms_output.put_line('No existe ese empleado');
END;
dbms_output.put_line('Continúo con la ejecución del bloque PL/SQL');
```

Ejemplo 2:

Crea un PL/SQL en el que dado el nif 99999999S de un empleado, incluya a ese empleado como recepcionista siempre que no existan ya más de 20 recepcionistas. Si no está dado de alta como empleado no se producirá error sino que se dará de alta tanto de empleado como de recepcionista.

```
DECLARE
      auxnombre empleado.nombre%type;
      total number(3);
      pasa_tope exception;
      elnif empleado.nif%type := '99999999S';
BEGIN
--Obtenemos los empleados que ya existen en recepción
 select count(*) into total from emp_recepcion;

 if total>20  then raise pasa_tope; end if;

--Vemos si ya existe como empleado

 select nombre  into auxnombre
 from empleado where nif=elnif;
--Como existe sólo insertamos en recepcionistas
 insert into emprecepcion values(elnif);
EXCEPTION
    WHEN no_data_found then
--no existe como empleado, lo insertamos en ambas tablas
        insert into empleado(nif) values(elnif);
        insert into emprecepcion values(elnif);
   WHEN pasa_tope then
      dbms_output.put_line('Se supera el tope asignado para recepcionistas');
END;
```