Sesión 5: MENSAJES.
DEFINICIÓN DE CONSTANTES, VARIABLES Y CURSORES.

Para mostrar mensajes podemos utilizar:

MENSAJES

raise_application_error (número_error, mensaje)
número_error –20000…-20999

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

Muestra el mensaje y genera un error, interrumpiendo la ejecución.

dbms_output.put_line(mensaje)

Deposita el mensaje en un buffer y prosigue con la ejecución

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

SET SERVEROUTPUT ON FORMAT

A la hora de componer los mensajes se utiliza el operador concatenación ||

Lo podemos usar para construir los mensajes tanto con raise_application_error como con dbms_ouput.put_line

BLOQUES SQL

Un bloque SQL tiene tres partes: una declarativa, una ejecutable y una última para el manejo de instrucciones (warnings y condiciones de error). De estas 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';
maximo CONSTANT number(4):=9999;
```

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**.

```
aux_valor1:=25;
aux_valor2:=total-10;
aux_nombre:='ESTEBAN'
```

```
SELECT categoria  INTO auxcategoria  FROM habitacion WHERE ….
SELECT cod, descripción  INTO auxcod, auxdesc  FROM ACTIVIDADES WHERE …
```

CURSOR

Oracle utiliza áreas de trabajo para ejecutar sentencias SQL y almacenar la información procesada.
El uso de cursores permite dar nombre a un área de trabajo y acceder a la información almacenada en ella.

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 *(aunque esta forma de trabajar con cursores es válida, recomendamos trabajar con cursores de un modo más sencillo, t*
con CURSORES en BUCLE FOR, que se explica en el siguiente apartado)

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

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.

```
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;
        …
        CLOSE c1;
        …
END;
```

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

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

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.

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:

Si c1 es un cursor que hemos definido, se podrían utilizar expresiones como

```
IF c1%FOUND THEN …
IF c1%ROWCOUNT >4 THEN …
```

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

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;
```

Suponemos creada MALCATEGORIA(num: number(3), dif: number(4,2))

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

En el bloque ejecutable (BEGIN-END) se puede abrir el cursor y trabajar con él.

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)

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)

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;
```