BASES DE DATOS B
UF3

Lenguaje SQL: DCL y extensión procedimental

Introducción

- PL/SQL → Procedural Language/Structured Query Language
- Lenguaje procedural diseñado para trabajar junto con SQL
- Incluido en Oracle Database Server
- Características:
  - Integrado con SQL
  - Control de errores y excepciones
  - Uso de variables
  - Estructuras de control de flujo
  - Soporta programación orientada a objetos
  - Programación modular
- Procedimientos
- Funciones

Bloques de código anónimos

- Es la forma más básica de programar en PL/SQL
- Son fragmentos de código que no se almacenan en la estructura de la BBDD
- Para ejecutarlos solo es necesario introducirlos en la consola como si de SQL se tratase.
- Su estructura básica es:

```
DECLARE
    --Declaración de variables, cursores, excepciones, etc
BEGIN
    --Cuerpo del programa, código SQL, estructuras de control,
EXCEPTION
    -- GESTION DE EXCEPCIONES. Define acciones a realizar cuando aparece una excepcion
END;
```

Tipos de datos

- PL/SQL es un lenguaje de programación fuertemente tipado, por lo que las variables deben tener un tipo definido.
- TIPOS:
  - NUMBER (Numérico) → Números enteros o decimales.
  - CHAR (Carácter) → Almacena caracteres, su longitud por defecto es 1
  - VARCHAR2 (Cadena de texto) → Almacena cadenas de caracteres
  - BOOLEAN (lógico) → TRUE/FALSE
  - DATE (FECHA)
  - %TYPE → Asignan a una variable el tipo de otra ya definida.
  - %ROWTYPE → Asigna a una variable un tipo fila de tabla, es decir, una variable capaz de almacenar todos los campos de una tabla.

Declaración de variables

- Se deben declarar en el apartado DECLARE
- Sintaxis: NombreVariable [CONSTANT] tipoVariable [NOT NULL]
- Ejemplos:
  - Precio NUMBER(6,2);
  - FechaNacimiento DATE;
  - Apellido VARCHAR2(50):='Martinez Salmaz';
  - Stock Productos.Cantidad%TYPE;
  - Producto Productos%ROWTYPE;

Estructuras de control

- PL/SQL, como la mayoría de lenguajes de programación, nos proporciona estructuras de control para la toma de decisiones y la iteración:
- Sentencias IF
- Se emplea para, en función de una expresión evaluada ejecutar unas instrucciones u otras.

```
IF condición THEN
intrucciones;
[ELSIF condición THEN
instrucciones;]
[ELSE
  instrucciones;]
ENDIF;
```

```
IF Altura>54 THEN
  Valido:=TRUE;
ELSIF Altura=23 THEN
  VALIDO:=FALSE;
ELSE
  DBMS_OUTPUT.PUT_LINE('ERRO
END;
```

Estructuras de control

- Sentencia CASE
- Sentencia de selección múltiple. Evalúa la expresión contra varias posibles opciones.

```
CASE Variable
          WHEN Valor THEN
          Instrucciones;
          WHEN Valor2 THEN
          Instrucciones
ELSE
          Instrucciones;
END CASE;
```

```
CASE Altura
   WHEN 1 THEN
     Valido:=TRUE;
   WHEN 2 THEN
     Valido:=2;
   ELSE
      DBMS_OUTPUT.PUT_LINE('ERROR');
END CASE;
```

Estructuras de repetición

- Sentencia LOOP
- Bucle infinito, independiente de la evaluación de la condición de salida

```
LOOP
    Instrucciones;
END LOOP;
```

```
LOOP
  Altura:=Altura+1;
  IF Altura>45 THEN
    EXIT;
  END IF;
END LOOP;
```

```
LOOP
  Altura:=Altura+1;
  EXIT WHEN Altura>45;
END LOOP;
```

Estructuras de repetición

- Sentencia WHILE
- Bucle que se ejecutará mientras la condición a evaluar se cumpla

```
WHILE Condición Loop
   Instrucciones;
END LOOP;
```

```
WHILE Altura<45 LOOP
   Altura:=Altura+1;
END LOOP;
```

Estructuras de repetición

- Sentencia FOR
- Usado cuando se conoce de antemano el número de repeticiones
- Bucle que permite declarar implícitamente un índice para

```
FOR Indice IN valorInicial .. valorFinal Loop
    Instrucciones;
END LOOP;
```

```
FOR i IN 0 .. 100 LOOP
  SYS.DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
```

Procedimientos

```
CREATE [OR REPLACE]
PROCEDURE <procedure_name> [(<param1> [IN|OUT|IN OUT] <type>,
                             <param2> [IN|OUT|IN OUT] <type>,
...)]
IS
 -- Declaración de variables locales
BEGIN
  -- Sentencias
[EXCEPTION]
  -- Sentencias control de excepcion
END [<procedure_name>];
```

- Un procedimiento es un subprograma que ejecuta una acción específica y que no devuelve ningún valor.
- Un procedimiento tiene un nombre, un conjunto de parámetros (opcional) y un bloque de código.

Procedimientos

```
create or replace procedure joc(i number) is
  num constant number := 2525;
begin
  case
     when i < num then
        dbms_output.put_line('El numero es mas pequeño, continua buscando...');
     when i > num then
        dbms_output.put_line('El numero es mas grande, continua buscando...');
     when i = num then
        dbms_output.put_line('HAS ACERTADO !!!!');
   end case;
end;
```

- En el ejemplo hemos creado un procedimiento que muestra los números desde el 1 hasta el valor pasado por parámetro.
- Para llamar al procedimiento una vez creado simplemente debemos poner su nombre y, entre paréntesis el parámetro o parámetros necesarios.

Parámetros

- Para declarar parámetros de un procedimiento debemos seguir la siguiente sintaxis:
- <param1> [IN | OUT | INOUT] <type>
  - **<param1>** → es el nombre del parámetro
  - **<type>** → es el tipo de dato
  - **[IN | OUT | INOUT]** → Indica la forma en la que pasamos el parámetro al procedimiento.
  - **IN:** El parámetro es de entrada, significa que la variable original (fuera de la función) no se vera afectada.
  - **OUT:** El parámetro es de salida. El parámetro se usará para almacenar un valor de salida del procedimiento. No se puede emplear como parámetro de entrada.
  - **IN OUT:** El parámetro actúa como parámetro de entrada/salida, es decir, se emplea para pasar un valor al procedimiento y, además, como forma de almacenar un valor de salida.

Funciones

- Las funciones PL/SQL son unidades funcionales similares a los procedimientos, la principal diferencia redica en que las funciones devuelven un resultado tras su ejecución.
- Es necesario indicar el tipo de dato que la función va a devolver en la definición de la misma.

```
CREATE [OR REPLACE]
FUNCTION <fn_name>[(<param1> IN <type>, <param2> IN <type>, ...)]
RETURN <return_type>
IS
  result <return_type>;
BEGIN

  return(result);
[EXCEPTION]
  -- Sentencias control de excepción
END [<fn_name>];
```

Funciones

- En el ejemplo hemos programado una función que recibe dos números por parámetro y devuelve la suma de ambos.

```
CREATE OR REPLACE
FUNCTION sumarNumeros(num1 number, num2 number)
RETURN NUMBER
IS
   resultado NUMBER;
BEGIN
   resultado:=num1+num2;
   return(resultado);
END ;
```

Funciones

- Para llamar a una función se hace de forma similar a como llamábamos a los procedimientos, sin embargo, ahora podemos usar el dato que nos devuelve la función asignando a una variable o bien imprimiendo por pantalla.

```
Begin
   sumarNumeros(2,2);
End;
```

```
Declare
   suma Number;
Begin
   suma:=sumarNumeros(2,2);
end;
```

```
Begin
  DBMS_OUTPUT.PUT_LINE(sumarNumeros(2,2));
end;
```

Excepciones

- Estas tienen que declarese dentro de su bloque (EXCEPTION)
- Las excepciones dentro de Oracle sirven para tratar errores en tiempo de ejecución.
- Cuando ocurre un error el sistema pasa el control al bloque de código de las excepciones (EXCEPTION)
- Estas pueden ser definidas por el sistema o por el usuario.

```
DECLARE
   ...
BEGIN
   ...
EXCEPTION
   WHEN [ERROR] THEN
        ...
END;
```

Como Excepciones definidas por el sistema encontramos varias, aunque las más relevantes son:

Excepciones propias de ORACLE

Nombre	Descripción
NO_DATA_FOUND	La sentencia *SELECT* no devuelve ningún valor.
TOO_MANY_ROWS	La sentencia *SELECT* devuelve más de una fila.
INVALID_CURSOR	Se está haciendo referencia a un cursor no válido.
ZERO_DIVIDE	Se está intentando realizar una división de un número entre cero.
CASE_NOT_FOUND	Ninguna de las condiciones de la sentencia *WHEN* en la estructura *CASE* se corresponde con el valor evaluado y no existe cláusula *ELSE*.
CURSOR_ALREADY_OPEN	El cursor que intenta abrirse ya está abierto.
INVALID_NUMBER VALUE_ERROR	La conversión de una cadena a valor numérico no es posible porque la cadena no representa un valor numérico válido.
VALUE_ERROR	Error ocurrido en alguna operación aritmética, de conversión o trucado.
LOGIN_DENIED	Un programa está intentado acceder a la base de datos con un usuario o *password* incorrecto.
NOT_LOGGED_ON	Un programa está intentado ejecutar una acción en la base de datos sin haber formalizado previamente la conexión.
TIMEOUT_ON_RESOURCE	Se ha acabado el tiempo que el SGBD pude esperar por algún recurso.
OTHERS	Es la opción por defecto. Interceptará todos los errores no tenidos en cuenta en las condiciones WHEN anteriores.

```
-- Ejemplo
DECLARE
 fecha_incorrecta EXCEPTION;
 PRAGMA EXCEPTION_INIT (fecha_incorrecta, -01847);
 fecha DATE;
BEGIN
 fecha := TO_DATE('32-12-2017');
 dbms_output.put_line(fecha);
 EXCEPTION
  WHEN fecha_incorrecta THEN
   dbms_output.put_line('La fecha introducida es incorrecta');
END;
```

```
-- Ejemplo
DECLARE
   negativo EXCEPTION;
   valor NUMBER;
BEGIN
    valor:= -1;
    IF valor < 0 THEN
        RAISE negativo;
    END IF;
EXCEPTION
    WHEN negativo THEN
      dbms_output.put_line('Números negativos NO permitidos');
END;
```

Excepciones definidas por el usuario

- Los usuarios pueden lanzar dos tipos de errores diferentes:
  - Error predefinido de Oracle.
  - Error propio del usuario.

Cursores

- Son utilizados para recorrer consultas, siendo un conjunto de registros devuelto por una consulta.
- Estos pse puede clasificar en dos tipos implícitos y explícitos.
- Su definición es de la misma forma que si fuera una variable.
- Estos pueden utilizar parámetros que se declaren junto a este y se combinaran con estructuras de repetición para mostrar los datos.

Cursores Implícitos

- Son aquellos que solo devuelven un único registro.
- Muy utilizados en sentencias INTO.

```
-- Ejemplo
SET SERVEROUTPUT ON;
DECLARE
vnombre VARCHAR2(50);
BEGIN
SELECT nombre INTO vnombre  FROM Alumno  WHERE codigo=14;
DBMS_OUTPUT.PUT_LINE('El nombre del alumno es : ' || vnombre);
END;
```

Cursores Explícitos

- Son aquellos que devuelven más de un registro.
- Almacenan en posiciones de memoria los resultados de sentencias SELECT que precisan de tratamiento "*fila a fila*".
- Son los más utilizados y muy rápidos.

```
-- Sintaxis

DECLARE  CURSOR nombre_cursor IS <consulta_SELECT> ;

BEGIN

OPEN nombre_cursor;  -- Abrir el cursor

FETCH nombre_cursor INTO VARIABLES_PLSQL  -- Recorrer el cursor

CLOSE nombre_cursor ;  -- Cerrar el cursor

END;
```

Triggers

- Los Triggers o disparadores son módulos asociados a una tabla que se ejecutaran con la realización de determinadas acciones en la tabla a la cual van atados.
- Hay que definir cuando serán ejecutados si antes o después de la acción en la tabla.
- Hay que definir también la acción en la tabla que lo disparara, la inserción de datos, la actualización, etc..

```
CREATE OR REPLACE TRIGGER nombre-del-trigger
[FOLLOWS nombre-otro-trigger]
[BEFORE/AFTER]
[INSERT/DELETE/UPDATE/UPDATE OF lista-columnas] ON nombre-tabla
[REFERENCING [OLD AS nombre-antiguo][NEW AS nombre-nuevo]]
[FOR EACH ROW/FOR EACH STATEMENT]
[WHEN {condiciones}]
{Bloque estándar de sentencias PL/SQL... BEGIN, EXCEPTION}
```

Triggers

- En este caso vemos un disparador que se ejecutará antes de la acción de insertar o actualizar el salario de una tabla de empleados.
- El FOR EACH ROW nos activara cada fila que este afectada por la sentencia lanzada, en este caso INSERT o UPDATE con la restricción opcional que podemos indicar en la clausula WHEN.
- Dentro de un bloque de código indicaremos las acciones a realizar cuando cumplimos la condición.

```
CREATE TRIGGER trl_empleados
BEFORE
INSERT OR UPDATE OF salario ON empleados
FOR EACH ROW WHEN (:new.salario > 5000)
BEGIN
    UPDATE empleados SET salario = 5000 WHERE empleado_id = :new.empleado_id;
END;
```

BASES DE DATOS B UF3

Bases de datos objetos relacionales