Bloque PL/SQL

Un bloque PL/SQL es una unidad de código que contiene una o más sentencias PL/SQL. La estructura general de un bloque PL/SQL es la siguiente:

Estructura general de un bloque

```
DECLARE
   -- Declar aciones de
variables locales y tipos de
datos person ali zados
BEGIN
   -- Sentencias PL/SQL
   -- Incluyendo sentencias
SELECT, INSERT, UPDATE, DELETE,
y llamadas a proced imi entos y
funciones
    EXC EPTION
   -- Manejo de errores
END;
```

Ejemplo

```
DECLARE
    res ultado NUMBER;
BEGIN
    res ultado := 10 + 5;
    DBM S_O UTP UT.P UT _LI -
NE('El resultado es: ' ||
result ado);
EXCEPTION
   WHEN OTHERS THEN
        DBM S_O UTP UT.P UT -
_LI NE( 'Error: ' || SQLERRM);
END;
```

Estructura general

DECLARE	se utiliza para declarar variables locales y tipos de datos personalizados que se utilizarán dentro del bloque.
BEGIN	se colocan las sentencias PL/SQL que realizan alguna operación.
EXCEPTION	se utiliza para manejar cualquier error que ocurra durante la ejecución del bloque PL/SQL.

Declaración variables

```
DECLARE
        <No mbr e> <Ti po> :=
<Va lor >;
DECLARE
        nombre VARCHA R2(50) :=
'Juan';
DECLARE
        es_ ver dadero BOOLEAN
:= TRUE;
DECLARE
       edad NUMBER:= 25;
```

Declaración constantes

```
DECLARE
        <No mbr e> CONSTANT
<Ti po> := <Va lor >;
DECLARE
        nombre CONSTANT VARCHA -
R2(50) := 'Juan';
DECLARE
        es_ ver dadero CONSTANT
BOOLEAN := TRUE;
DECLARE
       edad CONSTANT NUMBER:=
25;
```

Procesdimientos y funciones

Los procedimientos y las funciones son subprogramas que permiten agrupar y reutilizar bloques de código. Los procedimientos se utilizan para realizar acciones o tareas, mientras que las funciones devuelven un valor

Procedimientos

```
CREATE OR REPLACE PROCEDURE
imprimir_saludo(nombre IN
VARCHAR2) IS
        BEGIN
                DBM S_O UTP -
UT.P UT _LI NE( '¡Hola, ' ||
nombre || '!');
       END;
-- Uso del proced imiento
BEGIN
    imp rim ir_ sal udo ('J -
uan');
END;
```

Funciones

```
CREATE OR REPLACE FUNCTION
totalCustomers
RETURN number IS
      total number(2) := 0;
BEGIN
      SELECT count(*) into total
     FROM customers;

      RETURN total;
END;
```

Instrucciones de control

Las instrucciones de control en PL/SQL permiten controlar el flujo de ejecución de un programa, permitiendo tomar decisiones y repetir acciones según sea necesario.

IF-THEN-ELSE

Esta instrucción se utiliza para tomar decisiones basadas en una condición. Si la condición es verdadera, se ejecuta un bloque de código; de lo contrario, se ejecuta un bloque de código diferente.

Ejemplo

```
DECLARE
      salario number := 5000;
BEGIN
     IF salario > 5000 THEN
            dbm s_o utp ut.p -
ut _li ne( 'Sa lario alto');
     ELSE
            dbm s_o utp ut.p -
ut _li ne( 'Sa lario bajo');
     END IF;
END;
```

CASE

Esta instrucción se utiliza para tomar decisiones basadas en múltiples condiciones. Se pueden definir varios casos, cada uno con su propia condición y código a ejecutar.

Ejemplo

```
DECLARE
      dia _semana number := 2;
BEGIN
     CASE dia_semana
            WHEN 1 THEN dbms_o -
utp ut.p ut _li ne( 'Lu nes');
            WHEN 2 THEN dbms_o -
utp ut.p ut _li ne( 'Ma rtes');
            WHEN 3 THEN dbms_o -
utp ut.p ut _li ne( 'Mi érc -
oles');
            WHEN 4 THEN dbms_o -
utp ut.p ut _li ne( 'Ju eves');
            WHEN 5 THEN dbms_o -
utp ut.p ut _li ne( 'Vi ern -
es');
            ELSE dbms_o utp -
ut.p ut _li ne('Fin de semana');
     END CASE;
END;
```

FOR LOOP

Esta instrucción se utiliza para repetir una acción un número determinado de veces.

Ejemplo

```
DECLARE
     i number;
BEGIN
     FOR i IN 1..5 LOOP
            dbm s_o utp ut.p -
ut _li ne( 'It eración ' || i);
     END LOOP;
END;
```

WHILE LOOP

Esta instrucción se utiliza para repetir una acción mientras se cumple una condición.

Ejemplo

```
DECLARE
     i number := 1;
BEGIN
      WHILE i <= 5 LOOP
            dbm s_o utp ut.p -
ut _li ne( 'It eración ' || i);
           i := i + 1;
     END LOOP;
END;
```

Cursores

En SQL, un cursor es una estructura de control utilizada para recorrer y manipular filas de un resultado de consulta de forma secuencial. Los cursores proporcionan un mecanismo para procesar registros uno a uno y realizar operaciones específicas en cada uno de ellos.

Ejemplo

```
DECLARE
   -- Declar ación del cursor
    CURSOR c_empl eados IS
        SELECT salario FROM
Empleados;

   -- Variables auxiliares
    tot al_ sal arios NUMBER :=
0;
    con tador NUMBER := 0;
    pro med io_ sal arios
NUMBER;
```

Ejemplo (cont)

```
> BEGIN
  -- Abrir el cursor
  OPEN c_empleados;

  -- Recorrer el cursor y calcular el total de los salarios
  FOR empleado IN c_empleados LOOP
    total_salarios := total_salarios + empleado.salario;
    contador := contador + 1;
  END LOOP;

  -- Calcular el promedio de los salarios
  IF contador > 0 THEN
    promedio_salarios := total_salarios / contador;
    DBMS_OUTPUT.PUT_LINE('El promedio de los salarios es: ' || promedio_salarios);
  ELSE
    DBMS_OUTPUT.PUT_LINE('No se encontraron empleados');
  END IF;

  -- Cerrar el cursor
  CLOSE c_empleados;
END;
```

Trigger BEFORE

Se ejecuta antes de que se aplique una operación en la tabla. Puede utilizarse para validar o modificar los datos antes de que se realice la acción en la tabla.

Ejemplo

```
CREATE OR REPLACE TRIGGER before_insert_trigger
BEFORE INSERT ON mi_tabla
FOR EACH ROW
BEGIN
   IF :new.valor > 0 THEN
       -- Permitir la inserción
        NULL;
   ELSE
       -- Cancelar la inserción
        RAI SE_ APP LIC ATI - ON_ ERR OR( -20001, 'El valor debe ser mayor que cero.');
   END IF;
END;
```

Trigger AFTER

Un trigger AFTER se ejecuta después de que se haya aplicado una operación en la tabla. Se utiliza principalmente para realizar acciones posteriores a la modificación de datos.

Ejemplo

```
CREATE OR REPLACE TRIGGER after_update_trigger
AFTER UPDATE ON mi_tabla
FOR EACH ROW
BEGIN
    INSERT INTO histor ial - _tabla (id_re gistro, fecha_ - mod ifi cacion)
    VALUES (:old.id, SYSDATE);
END;
```

Ejemplo (cont)

```
> /
```

Trigger INSTEAD OF

Un trigger INSTEAD OF se utiliza en vistas actualizables y permite reemplazar la acción predeterminada que se llevaría a cabo en una operación de inserción, actualización o eliminación en la vista

Ejemplo

```
CREATE OR REPLACE TRIGGER instead_of_insert_trigger
INSTEAD OF INSERT ON mi_vista
BEGIN
    RAI SE_ APP LIC ATI ON_ - ERR OR( -20002, 'La vista no es actual iza ble.');
END;
/
```

Excepción

Evento que ocurre durante la ejecución de un bloque de código que interrumpe el flujo normal del programa. Las excepciones pueden ser causadas por errores de programación, condiciones imprevistas o errores del sistema.

Tipos

Predefinidas	ya están definidas en el lenguaje, como la excepción "-NO_DATA_FOUND" que se lanza cuando una consulta SELECT no devuelve ninguna fila.
Definidas por el usuario	son excepciones personalizadas creadas por el programador para manejar situaciones específicas en su código.
De sistema	son excepciones generadas por el sistema, como la excepción "ORA-00001: unique constraint violated" que se lanza cuando se intenta insertar una fila con una clave primaria duplicada.

Manejo de Excepciones

Se utiliza la estructura TRY-CATCH. El código que se sospecha que puede generar una excepción se coloca dentro del bloque TRY. Si se produce una excepción dentro del bloque TRY, se activa la excepción y el control se transfiere al bloque CATCH. Dentro del bloque CATCH, se puede proporcionar una respuesta a la excepción y tomar medidas para manejarla.

Ejemplo

```
DECLARE
    v_e mp_name VARCHA R2(50);
BEGIN
    SELECT emp_name INTO
v_emp_name FROM employees WHERE
emp_id = 1000;
    DBM S_O UTP UT.P UT _LI -
NE( 'Em ployee Name: ' ||
v_emp_ name);
EXCEPTION
   WHEN NO_DAT A_FOUND THEN
        DBM S_O UTP UT.P UT -
_LI NE('No employee found with
ID 1000');
END;
```

Triggers

Es un bloque de código que se ejecuta automáticamente en respuesta a ciertos eventos, como la inserción, actualización o eliminación de filas en una tabla.

Uso

Un trigger puede ser utilizado para realizar acciones específicas antes o después de que ocurra un evento, como validar datos, auditar cambios, actualizar valores relacionados, entre otras acciones.