

# PL/SQL Básico a Intermedio

GUIA DE ESTUDIO

MARTIN ROJAS

## INDICE

Bloque anónimo .....	7
Variables .....	7
Tipos de variables.....	8
Constantes .....	9
Suma de dos números enteros .....	10
Suma de dos números enteros, uno con valor null .....	10
Suma de dos números enteros más una constante .....	10
Concatenar nombre edad y fecha de nacimiento .....	11
Operadores.....	11
Operadores lógicos .....	12
Dato boolean .....	12
Atributos.....	12
Tipo %TYPE .....	12
Tipo %ROWTYPE.....	13
Práctica con variables, constantes y %TYPE .....	13
Bloques .....	14
Bloques anidados .....	14
Bloques anidados con variables.....	14
Bloques anidados con funciones .....	15
Prácticas de funciones .....	15
Práctica 1 .....	15
Práctica 2 .....	16
2.1.- Primera opción todo junto .....	16
2.2.- Segunda opción separado .....	16
Sentencia IF.....	17
Sentencia ELSIF .....	17
Práctica IF.....	18
Práctica 1 .....	18
Práctica 2 .....	19

CASE con PL/SQL .....	20
CHAR .....	20
NUMBER .....	20
VARCHAR2 .....	21
Shared case .....	21
Práctica CASE .....	22
Práctica 1 .....	22
Bucles .....	22
Bucle LOOP .....	22
Bucles anidados .....	23
Bucle CONTINUE .....	24
Bucle FOR ascendente .....	24
Bucle FOR descendente .....	25
Bucle WHILE .....	25
GO TO dentro de bucle FOR .....	26
Prácticas de bucles .....	27
Práctica 1 .....	27
Práctica 2 .....	28
Práctica 3 .....	29
Práctica 4 .....	29
Práctica 5 .....	30
Operaciones DML con PL/SQL .....	30
SELECT .....	30
INSERT .....	31
UPDATE .....	31
DELETE .....	31
Práctica SELECT INTO .....	32
Práctica 1 .....	32
Práctica 2 .....	32
Práctica 3 .....	32
Práctica 4 .....	33

Práctica INSERT UPDATE DELETE .....	33
Práctica 1 .....	33
Práctica 2 .....	34
Práctica 3 .....	34
Excepciones .....	35
Ejemplo .....	35
Excepciones pre-definidas .....	36
Capturar excepcion con variables .....	36
Controlar SQL con excepciones .....	37
Excepciones por el usuario .....	37
Ámbito de las excepciones .....	38
Comando para crear errores por el usuario .....	39
Ejemplo INSERT de fecha .....	39
Prácticas excepciones .....	40
Práctica 1 .....	40
Práctica 2 .....	40
Práctica 3 .....	41
Práctica 4 .....	41
Práctica 5 .....	42
Práctica 6 .....	43
Registros .....	43
INSERT, UPDATE con registros.....	44
Colecciones .....	45
Tipo simple.....	45
Tipo compuesto.....	45
Multiples compuestas.....	46
Práctica de colecciones y registros .....	47
Cursores.....	50
Implicitos .....	50
Explicitos .....	50
Recorrer un cursor con LOOP .....	51

Recorrer cursor con FOR .....	51
Recorrer cursor con subconsulta .....	52
Cursores con parámetros .....	52
Clausula WHERE CURRENT OF .....	52
Prácticas con cursores .....	53
Práctica 1 .....	53
Práctica 2 .....	53
Práctica 3 .....	54
Práctica 4 .....	55
Práctica 5 .....	55
Procedimientos.....	56
Crear un procedimiento .....	56
Llamar un procedimiento .....	56
Ver código fuente de procedimientos y funciones.....	56
Ejecutar procedimiento con parámetro IN .....	57
Crear procedimiento con parámetro IN .....	57
Crear procedimiento con parámetro OUT .....	58
Crear procedimiento con parámetros IN OUT.....	59
Funciones.....	60
Funciones con SQL .....	60
Prácticas con procedimientos y funciones .....	61
Práctica 1 .....	61
Práctica 2 .....	61
Práctica 3 .....	63
Práctica 4 .....	64
Práctica 5 .....	66
Paquetes .....	67
Crear un paquete.....	67
Crear el cuerpo de un paquete.....	67
Funciones con paquetes .....	69
Sobrecarga de procedimientos .....	70

UTL_FILE .....	70
Triggers .....	71
Crear un trigger .....	71
Before trigger .....	71
Trigger con eventos multiples.....	72
Ejemplo 1 .....	72
Ejemplo 2 .....	72
Controlar el evento .....	72
Controlar el evento columnas.....	73
Clausula WHEN .....	74
Comprobar estado de los triggers .....	74
Triggers en modo comando.....	75
Trigger compuesto .....	75
Triggers tipo DDL.....	76
OBJECT.....	77
Crear un objeto.....	77
Crear BODY del objeto .....	77
SELF .....	78
Métodos estáticos (globales) .....	79
Método constructor .....	81
Comprobar objetos.....	82
Sobrecarga de métodos .....	82
Herencia .....	84
Sobreescribir métodos.....	85
Crear columna tipo objeto .....	86
Crear una tabla con campo constraint check JSON.....	86
Almacenar datos tipo CLOB .....	88
Objeto JSON .....	89
JSON_EXISTS .....	89
JSON_VALUE .....	89
JSON_QUERY .....	90

JSON_TABLE.....	90
JSON_MERGEPATCH.....	90
JSON_TRANSFORM.....	91
JSON CON PL/SQL.....	92
Comando PUT en JSON con PL/SQL .....	93
Trabajar con la base de datos.....	94
Crear un ARRAY JSON.....	94
Documentos de un ARRAY.....	95
Funciones del ARRAY .....	96

## Bloque anónimo

```
SET SERVEROUTPUT ON
```

```
BEGIN
```

```
DBMS_OUTPUT.PUT_LINE('Martín'); --Impresión en pantalla
```

```
DBMS_OUTPUT.PUT_LINE('Rojas'); -- La paquetería DBMS siempre va dentro de
```

```
    -- un bloque anónimo y se concatena así: ||
```

```
NULL; -- Debe tener al menos un comando
```

```
END;
```

## Variables

- Debe comenzar por una letra.
- Podemos incluir números o letras.
- Puede tener caracteres especiales: \$ o subrayado.
- El nombre no puede ser más de 30 caracteres.
- No usar palabras reservadas.
- Las variables de PL/SQL se declaran e inicializan por la palabra DECLARE dentro de un bloque anónimo.
- Cuando se declara una variable se declara su tipo de dato y su longitud.
- Para inicializar una variable se hace con “:=” y termina con “;”
- Las variables creadas pueden pasarse como argumentos a otros bloques anónimos.
- Pueden almacenar valores de otros programas PL/SQL.

Ejemplo:

```
DECLARE
```

```
    SALARY NUMBER(2) := 1000;
```

```
    NAME VARCHAR2(100) := 'JOHN CONNORS';
```

```
    BIRTH_DATE DATE = '16-jun-1989';
```

```
BEGIN
```

```
    SALARY := SALARY*10;
```

```
    IF SALARY > 10000 THEN
```

```
        ETC.
```

```
END;
```



# Tipos de variables

Escalares

Compuestas

De referencia

LOBs Large Objects

Bind variables. Asociadas a los comandos SQL

Ejemplos en el curso:

CHAR(LONGITUD)

VARCHAR2(LONGITUD)

LONG

LONG RAW

NUMBER (PRESICIÓN, ESCALA)(DIGITOS, DECIMALES)

BINARY\_INTEGER

PLS\_INTEGER

BOOLEAN

BANARY\_FLOAT

BANARY\_DOUBLE

DATE

TIMESTAMP

TIMESTAMP WHIT TIME ZONE

TIMESTAMP WHIT LOCAL TIME ZONE

INTERVAL YEAR TO MONTH

INTERVAL YEAR TO SECOND

Ejemplo

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
NAME VARCHAR2(50);
```

```
LASTNAME VARCHAR2(50);
```

```
BEGIN
```

```
NAME:='JONH';
```

```
LASTNAME:='CONNORS';
```

```
DBMS_OUTPUT.PUT_LINE(NAME || ' ' || LASTNAME);
```

```
END;
```

## Constantes

Para darle formato al código CNTRL + F7

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
x CONSTANT NUMBER := 10;
```

```
z NUMBER NOT NULL := 20;
```

```
Y INTEGER(10) := 11;
```

```
BEGIN
```

```
dbms_output.put_line(x);
```

```
z := 30;
```

```
dbms_output.put_line(z);
```

```
dbms_output.put_line(Y);
```

```
END;
```

```
/
```

```
DECLARE
```

```
Y INTEGER(10);
```

```
BEGIN
```

```
dbms_output.put_line(Y);
```

```
END;
```

## Suma de dos números enteros

```
SET SERVEROUTPUT ON
DECLARE
  x NUMBER(2) := 5;
  y NUMBER(2) := 10;
  r NUMBER(2);
BEGIN
  r := x + y;
  dbms_output.put_line(r);
END;
```

## Suma de dos números enteros, uno con valor null

CUALQUIER OPERACIÓN CON NULL DEVUELVE UN NULL

```
SET SERVEROUTPUT ON
DECLARE
  x NUMBER(2) := 5;
  y NUMBER(2) := NULL;
  r NUMBER(2);
BEGIN
  r := x + y;
  dbms_output.put_line(r);
END;
```

## Suma de dos números enteros más una constante

```
SET SERVEROUTPUT ON
DECLARE
  x NUMBER(2) := 5;
  y NUMBER(2) := 10;
  z CONSTANT NUMBER := 2;
  r NUMBER(2);
BEGIN
  r := x + y + z;
  dbms_output.put_line(r);
END;
```

## Concatenar nombre edad y fecha de nacimiento

```
SET SERVEROUTPUT ON
DECLARE
  x VARCHAR2(20) := 'MI NOMBRE';
  y NUMBER(2) := 10;
  z DATE := '02-jun-2024';
BEGIN
  dbms_output.put_line(x);
  dbms_output.put_line(y);
  dbms_output.put_line(z);
  dbms_output.put_line(x || ' ' || y || ' ' || z);
END;
```

## Operadores

### OPERADORES MÁS HABITUALES

+ SUMA

- RESTA

/ DIVISIÓN

\* MULTIPLICACIÓN

\*\* EXPONENTE

|| CONCATENAR

```
SET SERVEROUTPUT ON
DECLARE
  X NUMBER :=5;
  Z NUMBER :=10;
  A VARCHAR2(50) := 'EXAMPLE';
  D DATE:='01-01-1990';
BEGIN
  DBMS_OUTPUT.PUT_LINE(SYSDATE);
  DBMS_OUTPUT.PUT_LINE(D+1);
END;
```

## Operadores lógicos

= IGUAL

<> DISTINTO DE

< MENOR QUE

> MAYOR QUE

>= MAYOR O IGUAL A

<= MENOR O IGUAL A

AND

NOT

OR

## Dato boolean

DECLARE

B1 BOOLEAN;

BEGIN

B1:=TRUE;

B1:=FALSE;

B1:=NULL;

END;

## Atributos

### Tipo %TYPE

DECLARE

X NUMBER;

Z X%TYPE; --SE DICE QUE Z ES DEL MISMO TIPO QUE DE X

EMPLE EMPLOYEES.SALARY%TYPE; -- EMPL ES DEL MISMO TIPO DE COLUMNA SALARY EN  
LA TABLA EMPLOYEE

BEGIN

EMPLE:=100;

END;

## Tipo %ROWTYPE

```
SET SERVEROUTPUT ON
DECLARE
    EMPLEADO EMPLOYEES%ROWTYPE; --CREA UNA VARIABLE DE TIPO TABLA
BEGIN
    SELECT --SOLO DEVUELVE UNA SOLA FILA
    * INTO EMPLEADO
    FROM
    EMPLOYEES
    WHERE
    EMPLOYEE_ID=100;
    DBMS_OUTPUT.PUT_LINE(EMPLEADO.FIRST_NAME||' '||EMPLEADO.SALARY);
END;
/
```

## Práctica con variables, constantes y %TYPE

```
SET SERVEROUTPUT ON
DECLARE
    y CONSTANT NUMBER := 21/100;
    p NUMBER(5, 2);
    r p%TYPE;
BEGIN
    p := 100;
    r := p * y;
    dbms_output.put_line(r);
END;
```

## Bloques

### Bloques anidados

```
SET SERVEROUTPUT ON
BEGIN
  dbms_output.put_line('ESTOY EN EL PRIMER BLOQUE');
  DECLARE
    x NUMBER := 10;
  BEGIN
    dbms_output.put_line(x);
  END;
END;
```

### Bloques anidados con variables

```
SET SERVEROUTPUT ON
DECLARE
X NUMBER := 10;
BEGIN
DBMS_OUTPUT.PUT_LINE('X:= '||X);
  DECLARE
    --X NUMBER := 20;
  BEGIN
    DBMS_OUTPUT.PUT_LINE('X:= '||X);
  END;
END;
```

## Bloques anidados con funciones

```
SET SERVEROUTPUT ON
DECLARE
X VARCHAR2(50);
MAYUS VARCHAR2(100);
FECHA DATE;
Z NUMBER:= 109.80;
BEGIN
X:= 'Ejemplo';
    DBMS_OUTPUT.PUT_LINE(SUBSTR(X,1,3)); --CADENA DE TEXTO, ORIGEN, CARACTERES A MOSTRAR
    MAYUS:=UPPER(X);
    DBMS_OUTPUT.PUT_LINE(MAYUS);
    FECHA:=SYSDATE;
    DBMS_OUTPUT.PUT_LINE(FECHA);
    DBMS_OUTPUT.PUT_LINE(FLOOR(Z));
END;
```

## Prácticas de funciones

### Práctica 1

```
SET SERVEROUTPUT ON -- Visualizar iniciales de un nombre
DECLARE
    N VARCHAR2(50);
    A1 VARCHAR2(50);
    A2 VARCHAR2(50);
    MAYUS_N VARCHAR2(50);
    MAYUS_A1 VARCHAR2(50);
    MAYUS_A2 VARCHAR2(50);
BEGIN
    N:= 'juan';
    A1:= 'gonzales';
    A2:= 'camarena';
    MAYUS_N:= UPPER(N);
    MAYUS_A1:= UPPER(A1);
    MAYUS_A2:= UPPER(A2);
    DBMS_OUTPUT.PUT_LINE(SUBSTR(MAYUS_N,1,1)||' '||
        SUBSTR(MAYUS_A1,1,1)||' '||
        SUBSTR(MAYUS_A2,1,1)); END;
```



## Práctica 2

Averiguar el día en que naciste

### 2.1.- Primera opción todo junto

```
SET SERVEROUTPUT ON
DECLARE
    FECHA DATE;
    NOMBRE_DIA VARCHAR2(50);
BEGIN
    FECHA:='21/01/1993';
    NOMBRE_DIA:= TO_CHAR(TO_DATE(FECHA, 'DD/MM/YYYY'), 'DAY');
    DBMS_OUTPUT.PUT_LINE(FECHA);
    DBMS_OUTPUT.PUT_LINE(NOMBRE_DIA);
END;

--SELECT TO_CHAR(TO_DATE('05-10-2023', 'DD-MM-YYYY'), 'DAY') AS NombreDelDia
--FROM dual;
```

### 2.2.- Segunda opción separado

```
SET SERVEROUTPUT ON
DECLARE
    FECHA DATE;
    NOMBRE_DIA VARCHAR2(50);
BEGIN
    FECHA:=TO_DATE('21/01/1993', 'DD/MM/YYYY');
    NOMBRE_DIA:= TO_CHAR(FECHA, 'DAY');
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(FECHA,'DD/MM/YYYY'));
    DBMS_OUTPUT.PUT_LINE(NOMBRE_DIA);
END;
```

## Sentencia IF

```
SET SERVEROUTPUT ON
DECLARE
    X NUMBER:= 20;
BEGIN
    --X:= 10;
    IF
        X=10
    THEN
        DBMS_OUTPUT.PUT_LINE('X:= '||X);
    ELSE
        DBMS_OUTPUT.PUT_LINE('X:= '||'OTRO VALOR');
    END IF;
END;
```

## Sentencia ELSIF

```
SET SERVEROUTPUT ON
DECLARE
    SALES NUMBER:= 5500;
    BONUS NUMBER:= 0;
BEGIN
    IF
        SALES > 5000 THEN
        BONUS := 900;
    ELSIF
        SALES > 4500 THEN
        BONUS := 500;
    ELSE
        BONUS := 100;
    END IF;
    DBMS_OUTPUT.PUT_LINE('SALES:= '||SALES||' '||'BONUS:= '||BONUS);
END;
```

# Práctica IF

## Práctica 1

SET SERVEROUTPUT ON -- NÚMEROS PARES Y NONES

DECLARE

  X NUMBER:= 1048406;

BEGIN

  IF

  MOD(X,2)=0 THEN

  DBMS\_OUTPUT.PUT\_LINE('X:= '||X||' ES PAR');

  ELSE

  DBMS\_OUTPUT.PUT\_LINE('X:= '||X||' ES IMPAR');

  END IF;

END;

--SELECT 1 AS NUMERO, MOD(1,2) AS RESIDUO FROM DUAL;

--SELECT 12 AS NUMERO, MOD(12,2) AS RESIDUO FROM DUAL;

## Práctica 2

```
SET SERVEROUTPUT ON
DECLARE
    TIPO_PRODUCTO CHAR(1):='E';
BEGIN
    IF
        TIPO_PRODUCTO = 'A' THEN
        DBMS_OUTPUT.PUT_LINE(TIPO_PRODUCTO||' '||' ELECTRONICA');
    ELSIF
        TIPO_PRODUCTO = 'B' THEN
        DBMS_OUTPUT.PUT_LINE(TIPO_PRODUCTO||' '||' INFORMATICA');
    ELSIF
        TIPO_PRODUCTO = 'C' THEN
        DBMS_OUTPUT.PUT_LINE(TIPO_PRODUCTO||' '||' ROPA');
    ELSIF
        TIPO_PRODUCTO = 'D' THEN
        DBMS_OUTPUT.PUT_LINE(TIPO_PRODUCTO||' '||' MUSICA');
    ELSIF
        TIPO_PRODUCTO = 'E' THEN
        DBMS_OUTPUT.PUT_LINE(TIPO_PRODUCTO||' '||' LIBROS');
    ELSE
        DBMS_OUTPUT.PUT_LINE('LETRA NO ASIGNADA');
    END IF;
END;
```

## CASE con PL/SQL

### CHAR

```
SET SERVEROUTPUT ON
DECLARE
    V1 CHAR(1);
BEGIN
    V1:='B';
    CASE V1
    WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('EXCELLENT');
    WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('VERY GOOD');
    WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('GOOD');
    WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('FAIR');
    WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('POOR');
    ELSE DBMS_OUTPUT.PUT_LINE('NO SUCH VALUE');
    END CASE;
END;
```

### NUMBER

```
SET SERVEROUTPUT ON
DECLARE
    V1 NUMBER;
BEGIN
    V1:='10';
    CASE V1
    WHEN '10' THEN DBMS_OUTPUT.PUT_LINE('EXCELLENT');
    WHEN '20' THEN DBMS_OUTPUT.PUT_LINE('VERY GOOD');
    WHEN '30' THEN DBMS_OUTPUT.PUT_LINE('GOOD');
    WHEN '40' THEN DBMS_OUTPUT.PUT_LINE('FAIR');
    WHEN '50' THEN DBMS_OUTPUT.PUT_LINE('POOR');
    ELSE DBMS_OUTPUT.PUT_LINE('NO SUCH VALUE');
    END CASE;
END;
```

## VARCHAR2

```
SET SERVEROUTPUT ON
DECLARE
    V1 VARCHAR2(50);
BEGIN
    V1:='VERDE';
    CASE V1
    WHEN 'VERDE' THEN DBMS_OUTPUT.PUT_LINE('EXCELLENT');
    WHEN 'ROJO' THEN DBMS_OUTPUT.PUT_LINE('VERY GOOD');
    WHEN 'AMARILLO' THEN DBMS_OUTPUT.PUT_LINE('GOOD');
    WHEN 'AZUL' THEN DBMS_OUTPUT.PUT_LINE('FAIR');
    WHEN 'MORADO' THEN DBMS_OUTPUT.PUT_LINE('POOR');
    ELSE DBMS_OUTPUT.PUT_LINE('NO SUCH VALUE');
    END CASE;
END;
```

## Shared case

```
SET SERVEROUTPUT ON
DECLARE
    BONUS NUMBER;
BEGIN
    BONUS := 100;
    CASE
    WHEN BONUS > 500 THEN DBMS_OUTPUT.PUT_LINE('EXCELLENT');
    WHEN BONUS <= 500 AND BONUS > 250 THEN DBMS_OUTPUT.PUT_LINE('VERY GOOD');
    WHEN BONUS <= 250 AND BONUS > 100 THEN DBMS_OUTPUT.PUT_LINE('GOOD');
    ELSE
    DBMS_OUTPUT.PUT_LINE('POOR');
    END CASE;
END;
```

# Práctica CASE

## Práctica 1

```
SET SERVEROUTPUT ON
DECLARE
    USUARIO VARCHAR2(40);
BEGIN
    USUARIO:= USER;
    CASE USER
    WHEN 'SYS' THEN DBMS_OUTPUT.PUT_LINE('Eres superadministrador');
    WHEN 'SYSTEM' THEN DBMS_OUTPUT.PUT_LINE('Eres un administrador normal');
    WHEN 'HR' THEN DBMS_OUTPUT.PUT_LINE('Eres de recursos humanos');
    ELSE DBMS_OUTPUT.PUT_LINE('Usuario no autorizado');
    END CASE;
END;
```

## Bucles

### Bucle LOOP

```
SET SERVEROUTPUT ON
DECLARE
    X NUMBER := 1;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE(X);
        X:=X+1;
        /*
        IF X = 11
        THEN EXIT;
        END IF;
        */
        EXIT WHEN X = 11;
    END LOOP;
END;
```

## Bucles anidados

```
SET SERVEROUTPUT ON
DECLARE
  S PLS_INTEGER:=0;
  I PLS_INTEGER:=0;
  J PLS_INTEGER;
BEGIN
  NULL;
  <<PARENT>>
  LOOP
    I:=I+1;
    J:=100;
    DBMS_OUTPUT.PUT_LINE('PARENT: '||I);
    <<CHILD>>
    LOOP
      EXIT PARENT WHEN (I>3);
      DBMS_OUTPUT.PUT_LINE('CHILD: '||J);
      J:=J+1;
      EXIT CHILD WHEN (J>105);
    END LOOP CHILD;
  END LOOP PARENT;
  DBMS_OUTPUT.PUT_LINE('FINISH CODE');
END;
```



## Bucle CONTINUE

```
SET SERVEROUTPUT ON
DECLARE
  X NUMBER:=0;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE('LOOP: X= '||TO_CHAR(X));
    X:= X+1;
    /*
    IF X<3
    THEN CONTINUE;
    END IF;
    */
    CONTINUE WHEN X<3;
    DBMS_OUTPUT.PUT_LINE('DESPUES DE CONTINUE: X= '||TO_CHAR(X));
    EXIT WHEN X=5;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('DESPUES DEL LOOP X= '||TO_CHAR(X));
END;
```

## Bucle FOR ascendente

```
SET SERVEROUTPUT ON
BEGIN
  FOR i IN 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE(i);
  END LOOP;
END;
```

## Bucle FOR descendente

```
SET SERVEROUTPUT ON
DECLARE
    i VARCHAR2(50):='aaaaa';
BEGIN
    FOR i IN REVERSE 5..15 LOOP
        DBMS_OUTPUT.PUT_LINE(i);
        EXIT WHEN i=10;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE(i);
END;
```

## Bucle WHILE

```
SET SERVEROUTPUT ON
DECLARE
    X NUMBER:=0;
    BANDERA BOOLEAN:=FALSE;
BEGIN
    WHILE X<10 LOOP
        DBMS_OUTPUT.PUT_LINE(X);
        X:=X+1;
        EXIT WHEN X=5;
    END LOOP;

    WHILE BANDERA LOOP
        DBMS_OUTPUT.PUT_LINE('LA BANDERA ES VERDADERA');
        BANDERA:=TRUE;
        EXIT;
    END LOOP;

    WHILE NOT BANDERA LOOP
        DBMS_OUTPUT.PUT_LINE('LA BANDERA ES FALSA');
        BANDERA:=TRUE;
    END LOOP;
END;
```

## GO TO dentro de bucle FOR

```
SET SERVEROUTPUT ON
DECLARE
  P VARCHAR2(50);
  N PLS_INTEGER:=4;
BEGIN
  FOR j IN 2..ROUND(SQRT(N)) LOOP
    IF N MOD j=0 THEN
      P:=' NO ES UN NÚMERO PRIMO';
      GOTO PRINT_NOW;
    END IF;
  END LOOP;
  P:= ' ES NÚMERO PRIMO';
  <<PRINT_NOW>>
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(N)||P);
END;
```

# Prácticas de bucles

## Práctica 1

SET SERVEROUTPUT ON –Tablas de multiplicar

DECLARE

X NUMBER;

Z NUMBER;

BEGIN

X:=1;

Z:=1;

LOOP

EXIT WHEN X=11;

DBMS\_OUTPUT.PUT\_LINE('TABLA DE MULTIPLICAR DEL :'||X);

LOOP

EXIT WHEN Z=11;

DBMS\_OUTPUT.PUT\_LINE(X\*Z);

Z:=Z+1;

END LOOP;

Z:=0;

X:=X+1;

END LOOP;

END;

SET SERVEROUTPUT ON

DECLARE

X NUMBER;

Z NUMBER;

BEGIN

X:=1;

Z:=1;

WHILE X<11 LOOP

DBMS\_OUTPUT.PUT\_LINE('Tabla de multiplicar del: '||X);

WHILE Z<11 LOOP

DBMS\_OUTPUT.PUT\_LINE(X\*Z);

Z:=Z+1;

END LOOP;

Z:=0;

X:=X+1;

END LOOP;

```

END;

SET SERVEROUTPUT ON
DECLARE
  X NUMBER;
  Z NUMBER;
BEGIN
  X:=1;
  Z:=1;
  FOR i IN 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE('Tabla de multiplicar del: '||i);
    FOR j IN 1..11 LOOP
      DBMS_OUTPUT.PUT_LINE(X*Z);
      Z:=Z+1;
    END LOOP;
    Z:=0;
    X:=X+1;
  END LOOP;
END;

```

## Práctica 2

```

SET SERVEROUTPUT ON
DECLARE
  TEXTO VARCHAR2(50);
  LIMITE NUMBER;
  CONTADOR NUMBER;
  FRASE_AL_REVES VARCHAR2(50);
BEGIN
  TEXTO:='FRASE';
  LIMITE:= LENGTH(TEXTO);
  WHILE LIMITE>0 LOOP
    FRASE_AL_REVES:=FRASE_AL_REVES||SUBSTR(TEXTO,LIMITE,1);
    LIMITE:=LIMITE-1;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE(FRASE_AL_REVES);
END;

```

### Práctica 3

```
SET SERVEROUTPUT ON
DECLARE
    TEXTO VARCHAR2(50);
    LIMITE NUMBER;
    CONTADOR NUMBER;
    FRASE_AL_REVES VARCHAR2(50);
BEGIN
    TEXTO:='FRXASE';
    LIMITE:= LENGTH(TEXTO);
    WHILE LIMITE>0 LOOP
        FRASE_AL_REVES:=FRASE_AL_REVES||SUBSTR(TEXTO,LIMITE,1);
        LIMITE:=LIMITE-1;
        EXIT WHEN UPPER(SUBSTR(TEXTO,LIMITE,1))='X';
    END LOOP;
    DBMS_OUTPUT.PUT_LINE(FRASE_AL_REVES);
END;
```

### Práctica 4

```
SET SERVEROUTPUT ON
DECLARE
    NOMBRE VARCHAR2(50);
    LIMITE NUMBER;
    A CHAR(1);
    RESULTADO VARCHAR2(50);
BEGIN
    NOMBRE:= 'XIOMARA';
    LIMITE:=LENGTH(NOMBRE);
    A:='*';
    FOR i IN 1..LIMITE LOOP
        RESULTADO:=RESULTADO||A;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE(NOMBRE||' --> '||RESULTADO);
END;
```

## Práctica 5

SET SERVEROUTPUT ON – Multiplo de cuatro

DECLARE

INICIO NUMBER;

FIN NUMBER;

BEGIN

INICIO:=1;

FIN:=20;

FOR i IN INICIO..FIN LOOP

IF MOD(i,4) = 0 THEN

DBMS\_OUTPUT.PUT\_LINE(i);

END IF;

END LOOP;

END;

## Operaciones DML con PL/SQL

### SELECT

SET SERVEROUTPUT ON

DECLARE

SALARIO NUMBER;

NOMBRE EMPLOYEES.FIRST\_NAME%TYPE; --CREA UNA VARIABLE DE TIPO COLUMNA DE UNA TABLA

BEGIN

SELECT --SOLO DEVUELVE UN SOLO VALOR Y UNA SOLA FILA

SALARY, FIRST\_NAME INTO SALARIO, NOMBRE

FROM

EMPLOYEES

WHERE

EMPLOYEE\_ID=100;

DBMS\_OUTPUT.PUT\_LINE(NOMBRE||' '||SALARIO);

END;

## INSERT

```
DECLARE
  COL1 TEST.C1%TYPE;
BEGIN
  COL1:=20;
  INSERT INTO TEST VALUES(COL1,'CCC');
  COMMIT;
END;
```

## UPDATE

```
DECLARE
  T TEST.C1%TYPE;
BEGIN
  T:=10;
  UPDATE TEST SET C2='BBB' WHERE C1=T;
  COMMIT;
END;
/
```

## DELETE

```
DECLARE
  T TEST.C1%TYPE;
BEGIN
  T:=20;
  DELETE FROM TEST WHERE C1=T;
  COMMIT;
END;
/
```



# Práctica SELECT INTO

## Práctica 1

```
SET SERVEROUTPUT ON
DECLARE
    SALARIO_MAXIMO NUMBER;
BEGIN
    SELECT
        MAX(SALARY) INTO SALARIO_MAXIMO
    FROM EMPLOYEES
    WHERE DEPARTMENT_ID=100;
    DBMS_OUTPUT.PUT_LINE(SALARIO_MAXIMO);
END;
```

## Práctica 2

```
SET SERVEROUTPUT ON
DECLARE
    NOMBRE VARCHAR2(50);
    PUESTO VARCHAR2(50);
BEGIN
    SELECT E.FIRST_NAME, D.DEPARTMENT_NAME INTO NOMBRE, PUESTO
    FROM EMPLOYEES E JOIN DEPARTMENTS D ON E.DEPARTMENT_ID = D.DEPARTMENT_ID
    WHERE E.EMPLOYEE_ID = 100;
    DBMS_OUTPUT.PUT_LINE(NOMBRE||' '||PUESTO);
END;
```

## Práctica 3

```
SET SERVEROUTPUT ON
DECLARE
    IDENTIFICADOR DEPARTMENTS.DEPARTMENT_ID%TYPE;
    NOMBRE VARCHAR2(50);
    NUMERO_EMPLEADOS NUMBER;
BEGIN
    IDENTIFICADOR:=10;
    SELECT D.DEPARTMENT_NAME, COUNT(*) INTO NOMBRE, NUMERO_EMPLEADOS
    FROM EMPLOYEES E JOIN DEPARTMENTS D ON E.DEPARTMENT_ID=D.DEPARTMENT_ID
    WHERE E.DEPARTMENT_ID = IDENTIFICADOR
    GROUP BY D.DEPARTMENT_NAME;
    DBMS_OUTPUT.PUT_LINE(NOMBRE||' '||NUMERO_EMPLEADOS);
END;
```

## Práctica 4

```
SET SERVEROUTPUT ON
DECLARE
  A NUMBER; --EMPLOYEES.SALARY%TYPE;
  B NUMBER; --EMPLOYEES.SALARY%TYPE;
  R NUMBER;
BEGIN
  SELECT MAX(SALARY) INTO A
  FROM EMPLOYEES;
  SELECT MIN(SALARY) INTO B
  FROM EMPLOYEES;
  R:=A-B;
  DBMS_OUTPUT.PUT_LINE('Salario máximo: '||A);
  DBMS_OUTPUT.PUT_LINE('Salario mínimo: '||B);
  DBMS_OUTPUT.PUT_LINE('Diferencia: '||R);
END;
```

## Práctica INSERT UPDATE DELETE

### Práctica 1

```
DECLARE
  D_ID DEPARTMENTS.DEPARTMENT_ID%TYPE;
  D_NAME DEPARTMENTS.DEPARTMENT_NAME%TYPE;
  M_ID DEPARTMENTS.MANAGER_ID%TYPE;
  L_ID DEPARTMENTS.LOCATION_ID%TYPE;
BEGIN
  SELECT
  MAX(DEPARTMENT_ID) INTO D_ID
  FROM DEPARTMENTS;
  D_ID:=D_ID+1;
  D_NAME:='Informática';
  M_ID:=100;
  L_ID:=1000;
  INSERT INTO DEPARTMENTS VALUES(D_ID,D_NAME,M_ID,L_ID);
  COMMIT;
END;
/
```

## Práctica 2

```
DECLARE
  L_ID DEPARTMENTS.LOCATION_ID%TYPE;
  D_ID DEPARTMENTS.DEPARTMENT_ID%TYPE;
BEGIN
  L_ID:=1700;
  D_ID:=271;
  UPDATE DEPARTMENTS SET LOCATION_ID=L_ID WHERE DEPARTMENT_ID=D_ID;
  COMMIT;
END;
/
```

## Práctica 3

```
DECLARE
  D_ID DEPARTMENTS.DEPARTMENT_ID%TYPE;
BEGIN
  D_ID:=271;
  DELETE FROM DEPARTMENTS WHERE DEPARTMENT_ID=D_ID;
  COMMIT;
END;
/
```

## Excepciones

```
DECLARE
BEGIN
EXCEPTION
--1.- NO_DATA_FOUND
--2.- TOO_MANY_ROWS
--3.- ZERO_DIVIDE
--4.- DUP_VAL_ON_INDEX
```

```
    WHEN EX1 THEN
        NULL;
    WHEN EX2 THEN
        NULL;
    WHEN OTHERS THEN
        NULL;
END;
/
```

## Ejemplo

```
SET SERVEROUTPUT ON
DECLARE
    SALARIO NUMBER;
    NOMBRE EMPLOYEES.FIRST_NAME%TYPE;
BEGIN
    SELECT
        SALARY, FIRST_NAME INTO SALARIO, NOMBRE
    FROM
        EMPLOYEES
    WHERE
        EMPLOYEE_ID>1;
    DBMS_OUTPUT.PUT_LINE(NOMBRE||' '||SALARIO);

EXCEPTION
--1.- NO_DATA_FOUND
--2.- TOO_MANY_ROWS
--3.- ZERO_DIVIDE
--4.- DUP_VAL_ON_INDEX

    WHEN NO_DATA_FOUND THEN
```

```

DBMS_OUTPUT.PUT_LINE('ERROR, EMPLEADO NO ENCONTRADO');
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE('ERROR, DEMASIADOS EMPLEADOS ENCONTRADOS');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('ERROR INDEFINIDO');
END;
/

```

### Excepciones pre-definidas

```

DECLARE
MI_EXCEP EXCEPTION;
PRAGMA EXCEPTION_INIT(MI_EXCEP,-937);
V1 NUMBER;
V2 NUMBER;
BEGIN
SELECT EMPLOYEE_ID,SUM(SALARY) INTO V1,V2 FROM EMPLOYEES;
DBMS_OUTPUT.PUT_LINE(V1);
EXCEPTION
WHEN MI_EXCEP THEN
DBMS_OUTPUT.PUT_LINE('FUNCION DE GRUPO INCORRECTA');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('ERROR INDEFINIDO');
END;
/

```

### Capturar excepcion con variables

```

SET SERVEROUTPUT ON
DECLARE
EMPLEADO EMPLOYEES%ROWTYPE; --CREA UNA VARIABLE DE TIPO TABLA
CODE NUMBER;
MSG VARCHAR2(100);
FECHA DATE;
BEGIN
SELECT --SOLO DEVUELVE UNA SOLA FILA
* INTO EMPLEADO
FROM
EMPLOYEES;
DBMS_OUTPUT.PUT_LINE(EMPLEADO.FIRST_NAME||' '||EMPLEADO.SALARY);
EXCEPTION
WHEN OTHERS THEN

```

```

CODE:=SQLCODE;
MSG:=SQLERRM;
FECHA:=SYSDATE;
DBMS_OUTPUT.PUT_LINE('ERROR: '||CODE||', '||MSG: '||MSG||', '||TO_CHAR(FECHA, 'DD/MM/YYYY
HH24:MI:SS')));
INSERT INTO ERRORS VALUES(CODE,MSG,FECHA);
END;
/

```

## Controlar SQL con excepciones

```

SET SERVEROUTPUT ON
DECLARE
    REG REGIONS%ROWTYPE;
    REG_CONTROL REGIONS.REGION_ID%TYPE;
BEGIN
    REG.REGION_ID:=100;
    REG.REGION_NAME:='AFRICA';
    SELECT REGION_ID INTO REG_CONTROL
    FROM REGIONS
    WHERE REGION_ID=REG.REGION_ID;
    DBMS_OUTPUT.PUT_LINE('LA REGION YA EXISTE.');
```

EXCEPTION

```

    WHEN NO_DATA_FOUND THEN
        INSERT INTO REGIONS VALUES (REG.REGION_ID,REG.REGION_NAME);
        COMMIT;
        DBMS_OUTPUT.PUT_LINE('REGION INSERTADA EXITOSAMENTE.');
```

END;

/

## Excepciones por el usuario

```

SET SERVEROUTPUT ON
DECLARE
    REG_MAX EXCEPTION;
    REGN NUMBER;
    REGT VARCHAR2(200);
BEGIN
    REGN:=99;
    REGT:='ASIA';
    IF REGN > 100 THEN
        RAISE REG_MAX;
```

```

ELSE
INSERT INTO REGIONS VALUES(REGN, REGT);
COMMIT;
DBMS_OUTPUT.PUT_LINE('REGION INSERTADA EXITOSAMENTE');
END IF;
EXCEPTION
WHEN REG_MAX THEN
DBMS_OUTPUT.PUT_LINE('LA REGION NO PUEDE SER MAYOR DE 100.');
```

WHEN OTHERS THEN

```

DBMS_OUTPUT.PUT_LINE('ERROR INDEFINIDO.');
```

END;

/

### Ámbito de las excepciones

```

SET SERVEROUTPUT ON
DECLARE
    REG NUMBER;
    REGT VARCHAR2(200);
BEGIN
    REG:=201;
    REGT:='SUDAFRICA';
    DECLARE
    CONTROL_REGIONES EXCEPTION;
    BEGIN
        IF REG > 200 THEN
            RAISE CONTROL_REGIONES;
        ELSE
            INSERT INTO REGIONS VALUES(REG, REGT);
            COMMIT;
            DBMS_OUTPUT.PUT_LINE('REGION INSERTADA EXITOSAMENTE');
```

END IF;

```

EXCEPTION
WHEN CONTROL_REGIONES THEN
DBMS_OUTPUT.PUT_LINE('LA REGION NO PUEDE SER MAYOR DE 200. Bloque hijo.');
```

END;

```

EXCEPTION
WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('ERROR INDEFINIDO.');
```

END;

## Comando para crear errores por el usuario

```
--RAISE_APPLICATION_ERROR -20000 Y -20999
SET SERVEROUTPUT ON
DECLARE
    REG_MAX EXCEPTION;
    REGN NUMBER;
    REGT VARCHAR2(200);
BEGIN
    REGN:=101;
    REGT:='ASIA';
    IF REGN > 100 THEN
        RAISE_APPLICATION_ERROR(-20001,'EL ID NO PUEDE SER MAYOR DE 100');
    ELSE
        INSERT INTO REGIONS VALUES(REGN, REGT);
        COMMIT;
        DBMS_OUTPUT.PUT_LINE('REGION INSERTADA EXITOSAMENTE');
    END IF;
END;
/
```

## Ejemplo INSERT de fecha

```
INSERT INTO ERRORS VALUES (10001, 'Error de ejemplo', TO_DATE('2023-10-05 14:30:15', 'YYYY-MM-DD
HH24:MI:SS'));
/
```



# Prácticas excepciones

## Práctica 1

```
SET SERVEROUTPUT ON
DECLARE
    EMP_ID NUMBER;
    NOMBRE VARCHAR2(50);
BEGIN
    EMP_ID := 1000;
    SELECT FIRST_NAME INTO NOMBRE
    FROM EMPLOYEES
    WHERE EMPLOYEE_ID = EMP_ID;
    DBMS_OUTPUT.PUT_LINE(NOMBRE);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Empleado inexistente');
END;
/
```

## Práctica 2

```
SET SERVEROUTPUT ON
DECLARE
    EMP_ID NUMBER;
    NOMBRE VARCHAR2(50);
BEGIN
    EMP_ID := 1000;
    SELECT FIRST_NAME INTO NOMBRE
    FROM EMPLOYEES
    WHERE EMPLOYEE_ID > 100;
    DBMS_OUTPUT.PUT_LINE(NOMBRE);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Empleado inexistente');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Demasiados empleados');
END;
/
```

### Práctica 3

```
SET SERVEROUTPUT ON
DECLARE
    SALARY NUMBER;
    EMP_ID NUMBER;
    CODE NUMBER;
    MSG VARCHAR2(100);
BEGIN
    EMP_ID := 100;
    SELECT SALARY INTO SALARY
    FROM EMPLOYEES
    WHERE EMPLOYEE_ID = EMP_ID;
    DBMS_OUTPUT.PUT_LINE(SALARY/0);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Empleado inexistente');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Demasiados empleados');
    WHEN OTHERS THEN
        CODE:=SQLCODE;
        MSG:=SQLERRM;
        DBMS_OUTPUT.PUT_LINE(CODE||' '||MSG);
END;
/
```

### Práctica 4

```
SET SERVEROUTPUT ON
DECLARE
    DUPLICADO EXCEPTION;
    PRAGMA EXCEPTION_INIT(DUPLICADO,-00001);
BEGIN
    INSERT INTO REGIONS VALUES(1,'MALASIA');
EXCEPTION
    WHEN DUPLICADO THEN
        DBMS_OUTPUT.PUT_LINE('Clave duplicada intente otra');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('ERROR INDEFINIDO');
END;
```

## Práctica 5

```
SET SERVEROUTPUT ON
DECLARE
    CONTROL_REGIONES EXCEPTION;
    REG NUMBER;
    REGT VARCHAR2(200);
BEGIN
    REG:=201;
    REGT:='SUDAFRICA';
    IF REG > 200 THEN
        RAISE CONTROL_REGIONES;
    ELSE
        INSERT INTO REGIONS VALUES(REG, REGT);
        COMMIT;
        DBMS_OUTPUT.PUT_LINE('REGION INSERTADA EXITOSAMENTE');
    END IF;
EXCEPTION
    WHEN CONTROL_REGIONES THEN
        DBMS_OUTPUT.PUT_LINE('LA REGION NO PUEDE SER MAYOR DE 200.');
```

WHEN OTHERS THEN

```
    DBMS_OUTPUT.PUT_LINE('ERROR INDEFINIDO.');
```

END;

/

## Práctica 6

```
SET SERVEROUTPUT ON
DECLARE
    REG NUMBER;
    REGT VARCHAR2(200);
BEGIN
    REG:=201;
    REGT:='SUDAFRICA';
    BEGIN
        IF REG > 200 THEN
            RAISE_APPLICATION_ERROR(-20001,'EL ID NO PUEDE SER MAYOR DE 200');
        ELSE
            INSERT INTO REGIONS VALUES(REG, REGT);
            COMMIT;
            DBMS_OUTPUT.PUT_LINE('REGION INSERTADA EXITOSAMENTE');
        END IF;
    END;
END;
/
```

## Registros

```
SET SERVEROUTPUT ON
DECLARE
    TYPE EMPLEADO IS RECORD(NOMBRE VARCHAR2(50),
        SALARIO NUMBER,
        FECHA EMPLOYEES.HIRE_DATE%TYPE,
        DATOS EMPLOYEES%ROWTYPE);

    EMPLE1 EMPLEADO;

BEGIN
    SELECT * INTO EMPLE1.DATOS
    FROM EMPLOYEES WHERE EMPLOYEE_ID=100;

    EMPLE1.NOMBRE:=EMPLE1.DATOS.FIRST_NAME||' '||EMPLE1.DATOS.LAST_NAME;
    EMPLE1.SALARIO:=EMPLE1.DATOS.SALARY*0.80;
    EMPLE1.FECHA:=EMPLE1.DATOS.HIRE_DATE;

    DBMS_OUTPUT.PUT_LINE(EMPLE1.NOMBRE);
```

```
DBMS_OUTPUT.PUT_LINE(EMPLE1.SALARIO);
DBMS_OUTPUT.PUT_LINE(EMPLE1.FECHA);
DBMS_OUTPUT.PUT_LINE(EMPLE1.DATOS.FIRST_NAME);
END;
/
```

### INSERT, UPDATE con registros

```
CREATE TABLE REGIONES AS SELECT * FROM REGIONS WHERE REGION_ID=0;
/
```

```
DECLARE
    REG1 REGIONS%ROWTYPE;
BEGIN
    SELECT * INTO REG1
    FROM REGIONS WHERE REGION_ID=1;
    --INSERT
    INSERT INTO REGIONES VALUES REG1;
END;
/
```

```
DECLARE
    REG1 REGIONS%ROWTYPE;
BEGIN
    REG1.REGION_ID:=1;
    REG1.REGION_NAME:='AUSTRALIA';
    --UPDATE
    UPDATE REGIONES SET ROW=REG1 WHERE REGION_ID=1;
END;
/
```

## Colecciones

### Tipo simple

```
SET SERVEROUTPUT ON
DECLARE
    TYPE DEPARTAMENTOS IS TABLE OF
        DEPARTMENTS.DEPARTMENT_NAME%TYPE
        INDEX BY PLS_INTEGER;
    DEPTS DEPARTAMENTOS;
BEGIN
    DEPTS(1):= 'INFORMATICA';
    DEPTS(2):= 'RH';
    DBMS_OUTPUT.PUT_LINE(DEPTS(1));
    DBMS_OUTPUT.PUT_LINE(DEPTS(2));
    DBMS_OUTPUT.PUT_LINE(DEPTS.LAST);
    DBMS_OUTPUT.PUT_LINE(DEPTS.FIRST);
    IF DEPTS.EXISTS(3) THEN
        DBMS_OUTPUT.PUT_LINE(DEPTS(3));
    ELSE
        DBMS_OUTPUT.PUT_LINE('NO EXISTE EL ÍNDICE');
    END IF;
END;
/
```

### Tipo compuesto

```
SET SERVEROUTPUT ON
DECLARE
    TYPE EMPLEADOS IS TABLE OF
        EMPLOYEES%ROWTYPE
        INDEX BY PLS_INTEGER;
    EMPLS EMPLEADOS;
BEGIN
    SELECT * INTO EMPLS(1)
    FROM EMPLOYEES
    WHERE EMPLOYEE_ID=100;
    DBMS_OUTPUT.PUT_LINE(EMPLS(1).FIRST_NAME);
END;
/
```

## Multiples compuestas

```
SET SERVEROUTPUT ON
DECLARE
  TYPE DEPARTAMENTOS IS TABLE OF
    DEPARTMENTS%ROWTYPE
    INDEX BY PLS_INTEGER;
  DEPTS DEPARTAMENTOS;
BEGIN
  FOR I IN 1..10 LOOP
    SELECT * INTO DEPTS(I)
    FROM DEPARTMENTS
    WHERE DEPARTMENT_ID=I*10;
  END LOOP;

  FOR i IN DEPTS.FIRST..DEPTS.LAST LOOP
    DBMS_OUTPUT.PUT_LINE(DEPTS(i).DEPARTMENT_NAME);
  END LOOP;
END;
/
```

## Práctica de colecciones y registros

```
SET SERVEROUTPUT ON
DECLARE
    TYPE EMPL_RECORD IS RECORD (NAME VARCHAR2(100),
                                SAL EMPLOYEES.SALARY%TYPE,
                                COD_DEPT EMPLOYEES.DEPARTMENT_ID%TYPE);
    TYPE EMPL_TABLE IS TABLE OF
        EMPL_RECORD
        INDEX BY PLS_INTEGER;
    EMPLEADOS EMPL_TABLE;
BEGIN
    FOR I IN 100..206 LOOP
        SELECT FIRST_NAME||' '||LAST_NAME, SALARY, DEPARTMENT_ID INTO EMPLEADOS(I)
        FROM EMPLOYEES
        WHERE EMPLOYEE_ID=I;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('TODA LA COLECCIÓN');
    FOR I IN EMPLEADOS.FIRST..EMPLEADOS.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(EMPLEADOS(I).NAME||' '||EMPLEADOS(I).SAL||'
'||EMPLEADOS(I).COD_DEPT);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('PRIMER EMPLEADO');
    DBMS_OUTPUT.PUT_LINE(EMPLEADOS(EMPLEADOS.FIRST).NAME);
    DBMS_OUTPUT.PUT_LINE('ULTIMO EMPLEADO');
    DBMS_OUTPUT.PUT_LINE(EMPLEADOS(EMPLEADOS.LAST).NAME);
    DBMS_OUTPUT.PUT_LINE('TODOS LOS EMPLEADOS ANTES');
    DBMS_OUTPUT.PUT_LINE(EMPLEADOS.COUNT);

    FOR I IN EMPLEADOS.FIRST..EMPLEADOS.LAST LOOP
        IF EMPLEADOS(I).SAL < 7000 THEN
            EMPLEADOS.DELETE(I);
        END IF;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('TODOS LOS EMPLEADOS DESPUES');
    DBMS_OUTPUT.PUT_LINE(EMPLEADOS.COUNT);

    DBMS_OUTPUT.PUT_LINE('PRIMER EMPLEADO');
    DBMS_OUTPUT.PUT_LINE(EMPLEADOS(EMPLEADOS.FIRST).NAME);
```



```

DBMS_OUTPUT.PUT_LINE('ULTIMO EMPLEADO');
DBMS_OUTPUT.PUT_LINE(EMPLEADOS(EMPLEADOS.LAST).NAME);

DBMS_OUTPUT.PUT_LINE('VOLVEMOS A VISUALIZAR LA COLECCIÓN');
FOR I IN EMPLEADOS.FIRST..EMPLEADOS.LAST LOOP
    null;--DBMS_OUTPUT.PUT_LINE(EMPLEADOS(I).NAME||' '||EMPLEADOS(I).SAL||'
'||EMPLEADOS(I).COD_DEPT);
    END LOOP;
END;
/

SET SERVEROUTPUT ON
DECLARE
TYPE EMPL_RECORD IS RECORD
(
    NAME VARCHAR2(100),
    SAL EMPLOYEES.SALARY%TYPE,
    COD_DEPT EMPLOYEES.DEPARTMENT_ID%TYPE);
TYPE EMPL_TABLE IS TABLE OF
    EMPL_RECORD
INDEX BY PLS_INTEGER;
EMPL EMPL_TABLE;
BEGIN
FOR I in 100..206 LOOP
SELECT FIRST_NAME||' '||LAST_NAME, SALARY, DEPARTMENT_ID INTO EMPL(I) FROM EMPLOYEES
WHERE EMPLOYEE_ID=I;
END LOOP;
FOR I IN EMPL.FIRST..EMPL.LAST LOOP
DBMS_OUTPUT.PUT_LINE(EMPL(I).NAME||' '||EMPL(I).SAL||' '||EMPL(I).COD_DEPT);
END LOOP;
DBMS_OUTPUT.PUT_LINE('EL PRIMERO');
DBMS_OUTPUT.PUT_LINE(EMPL(EMPL.FIRST).NAME);
DBMS_OUTPUT.PUT_LINE('EL ÚLTIMO');
DBMS_OUTPUT.PUT_LINE(EMPL(EMPL.LAST).NAME);
DBMS_OUTPUT.PUT_LINE('BORRAMOS LOS EMPLEADOS QUE GANEN MENOS DE 7000');
DBMS_OUTPUT.PUT_LINE('ANTES');
DBMS_OUTPUT.PUT_LINE(EMPL.COUNT);
FOR I IN EMPL.FIRST..EMPL.LAST LOOP
IF EMPL(I).SAL < 7000 THEN
    EMPL.DELETE(I) ;
END IF;

```

```
END LOOP;  
DBMS_OUTPUT.PUT_LINE('DESPUES');  
DBMS_OUTPUT.PUT_LINE(EMPL.COUNT);  
END;  
/
```

# Cursores

## Implicitos

```
--SON LOS QUE VIENEN CUANDO SE EJECUTA UN SELECT
--LOS IMPLICITOS FUNCIONAN SOBRE LOS ULTIMOS SQL EJECUTADOS
--SQL%ISOPEN  PREGUNTA SI EL CURSOR ESTA ABIERTO
--SQL%FOUND
--SQL%NOTFOUND
--SQL%ROWCOUNT

SET SERVEROUTPUT ON
BEGIN
    UPDATE TEST SET C2='PPPP' WHERE C1=100;
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
    IF SQL%FOUND THEN
        DBMS_OUTPUT.PUT_LINE('ENCONTRADO');
    END IF;
    IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('NO ENCONTRADO');
    END IF;
END;
/
```

## Explicitos

```
--SON LOS QUE SE ABREN POR EL USUARIO
--C1%ISOPEN  PREGUNTA SI EL CURSOR ESTA ABIERTO
--C1%FOUND
--C1%NOTFOUND
--C1%ROWCOUNT

--COMANDO PARA LEER FILA POR FILA UN CURSOR

SET SERVEROUTPUT ON
DECLARE
    CURSOR C1 IS SELECT * FROM REGIONS;
    V1 REGIONS%ROWTYPE;
BEGIN
```

```
OPEN C1;
FETCH C1 INTO V1;
DBMS_OUTPUT.PUT_LINE(V1.REGION_NAME);
FETCH C1 INTO V1;
DBMS_OUTPUT.PUT_LINE(V1.REGION_NAME);
CLOSE C1;
END;
/
```

### Recorrer un cursor con LOOP

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR C1 IS SELECT * FROM REGIONS;
  V1 REGIONS%ROWTYPE;
BEGIN
  OPEN C1;
  LOOP
    FETCH C1 INTO V1;
    EXIT WHEN C1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(V1.REGION_NAME);
  END LOOP;
  CLOSE C1;
END;
/
```

### Recorrer cursor con FOR

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR C1 IS SELECT * FROM REGIONS;
BEGIN
  FOR i IN C1 LOOP
    DBMS_OUTPUT.PUT_LINE(i.REGION_NAME);
  END LOOP;
END;
/
```

## Recorrer cursor con subconsulta

```
BEGIN
  FOR i IN (SELECT * FROM REGIONS) LOOP
    DBMS_OUTPUT.PUT_LINE(i.REGION_NAME);
  END LOOP;
END;
/
```

## Cursores con parámetros

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR C1(SAL NUMBER) IS SELECT * FROM EMPLOYEES
  WHERE SALARY > SAL;
  EMPL1 EMPLOYEES%ROWTYPE;
BEGIN
  OPEN C1(10000);
  LOOP
    FETCH C1 INTO EMPL1;
    EXIT WHEN C1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(EMPL1.FIRST_NAME||' '||EMPL1.SALARY);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('Encontre '||C1%ROWCOUNT||' empleados.');
```

```
  CLOSE C1;
END;
/
```

## Clausula WHERE CURRENT OF

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR CUR IS SELECT * FROM EMPLOYEES FOR UPDATE;
  EMPL1 EMPLOYEES%ROWTYPE;
BEGIN
  OPEN CUR;
  LOOP
    FETCH CUR INTO EMPL1;
    EXIT WHEN CUR%NOTFOUND;
    IF EMPL1.COMMISSION_PCT IS NOT NULL THEN
      UPDATE EMPLOYEES SET SALARY = SALARY*1.10 WHERE CURRENT OF CUR;
    ELSE
      UPDATE EMPLOYEES SET SALARY = SALARY*1.15 WHERE CURRENT OF CUR;
```

```

        END IF;
    END LOOP;
CLOSE CUR;
END;
/

```

## Prácticas con cursores

### Práctica 1

```

SET SERVEROUTPUT ON
DECLARE
    NOMBRE VARCHAR2(100);
    SALARIO NUMBER;
BEGIN
    FOR i IN (SELECT * FROM EMPLOYEES) LOOP
        NOMBRE:=i.FIRST_NAME||' '||i.LAST_NAME;
        SALARIO:=i.SALARY;
        IF NOMBRE='Steven King' THEN
            RAISE_APPLICATION_ERROR(-20001,'EL SUELDO DEL JEFE NO SE PUEDE VER');
        ELSE
            DBMS_OUTPUT.PUT_LINE(NOMBRE||' '||SALARIO);
        END IF;
    END LOOP;
END;

```

### Práctica 2

```

SET SERVEROUTPUT ON
DECLARE
    M_ID NUMBER;
    NOMBRE VARCHAR2(100);
    NOMBRE_DEPARTAMENTO VARCHAR2(100);
BEGIN
    FOR i IN (SELECT * FROM EMPLOYEES) LOOP
        NOMBRE:=i.FIRST_NAME||' '||i.LAST_NAME;
        M_ID:=i.MANAGER_ID;
        IF i.MANAGER_ID <> 0 THEN
            FOR j IN (SELECT * FROM DEPARTMENTS WHERE MANAGER_ID=M_ID)LOOP
                NOMBRE_DEPARTAMENTO:=j.DEPARTMENT_NAME;
            END LOOP;
            DBMS_OUTPUT.PUT_LINE(NOMBRE||' jefe del departamento: '||NOMBRE_DEPARTAMENTO||');
        END IF;
    END LOOP;
END;

```

```

ELSE
DBMS_OUTPUT.PUT_LINE(NOMBRE||' no es jefe. ');
END IF;
END LOOP;
END;
/

SET SERVEROUTPUT ON
DECLARE
CURSOR C1 IS SELECT * FROM EMPLOYEES;
CURSOR C2(j DEPARTMENTS.MANAGER_ID%TYPE) IS SELECT * FROM DEPARTMENTS
WHERE MANAGER_ID=j;
DEPARTAMENTO DEPARTMENTS%ROWTYPE;
jefe DEPARTMENTS.MANAGER_ID%TYPE;
BEGIN
FOR EMPLEADO IN C1 LOOP
OPEN C2(EMPLEADO.EMPLOYEE_ID);
FETCH C2 INTO DEPARTAMENTO;
IF C2%NOTFOUND THEN
DBMS_OUTPUT.PUT_LINE(EMPLEADO.FIRST_NAME || ' No es JEFE de NADA');
ELSE
DBMS_OUTPUT.PUT_LINE(EMPLEADO.FIRST_NAME || ' ES JEFE DEL DEPARTAMENTO ' ||
DEPARTAMENTO.DEPARTMENT_NAME);
END IF;
CLOSE C2;
END LOOP;
END;
/

```

### Práctica 3

```

SET SERVEROUTPUT ON
DECLARE
CURSOR C1(COD DEPARTMENTS.DEPARTMENT_ID%TYPE ) IS SELECT COUNT(*) FROM EMPLOYEES
WHERE DEPARTMENT_ID=COD;
CODIGO DEPARTMENTS.DEPARTMENT_ID%TYPE;
NUM_EMPL NUMBER;
BEGIN
CODIGO:=80;
OPEN C1(CODIGO);
FETCH C1 INTO NUM_EMPL;
DBMS_OUTPUT.PUT_LINE('Número de empleados de ' ||CODIGO||' es '||NUM_EMPL);

```

```
CLOSE C1;  
END;  
/
```

## Práctica 4

```
SET SERVEROUTPUT ON  
DECLARE  
    NOMBRE VARCHAR2(100);  
BEGIN  
    FOR i IN (SELECT * FROM EMPLOYEES WHERE JOB_ID='ST_CLERK') LOOP  
        NOMBRE:=i.FIRST_NAME||' '||i.LAST_NAME;  
        DBMS_OUTPUT.PUT_LINE(NOMBRE);  
    END LOOP;  
END;  
/
```

## Práctica 5

```
SET SERVEROUTPUT ON  
DECLARE  
    CURSOR CUR IS SELECT * FROM EMPLOYEES FOR UPDATE;  
BEGIN  
    FOR EMPLE1 IN CUR LOOP  
        IF EMPLE1.SALARY > 8000 THEN  
            UPDATE EMPLOYEES SET SALARY = SALARY*1.02 WHERE CURRENT OF CUR;  
        ELSIF EMPLE1.SALARY < 8000 THEN  
            UPDATE EMPLOYEES SET SALARY = SALARY*1.03 WHERE CURRENT OF CUR;  
        END IF;  
    END LOOP;  
    COMMIT;  
END;  
/
```



## Procedimientos

### Crear un procedimiento

```
CREATE PROCEDURE P1
AS
    X NUMBER:=10;
BEGIN
    DBMS_OUTPUT.PUT_LINE(X);
END;
/
```

### Llamar un procedimiento

```
SET SERVEROUTPUT ON
BEGIN
    P1;
END;
/

EXECUTE P1;
/
```

### Ver código fuente de procedimientos y funciones

```
--USER OBJECTS
SELECT * FROM USER_OBJECTS
WHERE OBJECT_TYPE='PROCEDURE';
/

--OBJECTS
SELECT OBJECT_TYPE, COUNT(*) FROM USER_OBJECTS
GROUP BY OBJECT_TYPE;
/

--VER CÓDIGO FUENTE
SELECT * FROM USER_SOURCE
WHERE NAME='P1';
/
```

## Ejecutar procedimiento con parámetro IN

```
SET SERVEROUTPUT ON
DECLARE
  PARAM1 NUMBER;
BEGIN
  PARAM1 := 15;
  P2(PARAM1 => PARAM1); --(PARÁMETRO DE P2 => ASOCIADO A PARAM1 DEL BLOQUE ANÓNIMO)
--rollback;
END;
/
```

## Crear procedimiento con parámetro IN

```
CREATE OR REPLACE PROCEDURE CALC_TAX (EMPL IN EMPLOYEES.EMPLOYEE_ID%TYPE,
                                     T1 IN NUMBER)
AS
  TAX NUMBER:=0;
  SAL NUMBER:=0;
  EXEC_USER EXCEPTION;
BEGIN
  IF T1<0 OR T1>60 THEN
    RAISE EXEC_USER;
  ELSE
    SELECT SALARY INTO SAL FROM EMPLOYEES WHERE EMPLOYEE_ID=EMPL;
    TAX:=SAL*T1/100;
  END IF;
  DBMS_OUTPUT.PUT_LINE('SALARY: '||SAL);
  DBMS_OUTPUT.PUT_LINE('TAX: '||TAX);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('NO EXISTE EL EMPLEADO.');
```

```
  WHEN EXEC_USER THEN
    DBMS_OUTPUT.PUT_LINE('TAX DEBE SER ENTRE 0 Y 60.');
```

```
END;
/
SET SERVEROUTPUT ON
BEGIN
  CALC_TAX(100,70);
END;
/
EXECUTE CALC_TAX(100,70);
```

/

## Crear procedimiento con parámetro OUT

```
CREATE PROCEDURE CALC_TAX_OUT (EMPL IN EMPLOYEES.EMPLOYEE_ID%TYPE,  
    T1 IN NUMBER,  
    R1 OUT NUMBER)
```

```
AS
```

```
    SAL NUMBER:=0;
```

```
    EXEC_USER EXCEPTION;
```

```
BEGIN
```

```
    IF T1<0 OR T1>60 THEN
```

```
        RAISE EXEC_USER;
```

```
    ELSE
```

```
        SELECT SALARY INTO SAL FROM EMPLOYEES WHERE EMPLOYEE_ID=EMPL;
```

```
        R1:=SAL*T1/100;
```

```
    END IF;
```

```
    DBMS_OUTPUT.PUT_LINE('SALARY: '||SAL);
```

```
EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN
```

```
        DBMS_OUTPUT.PUT_LINE('NO EXISTE EL EMPLEADO.');
```

```
    WHEN EXEC_USER THEN
```

```
        DBMS_OUTPUT.PUT_LINE('TAX DEBE SER ENTRE 0 Y 60.');
```

```
END;
```

/

--FORSOZAMENTE DE EJECUTAN ASI:

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
    A NUMBER;
```

```
    B NUMBER;
```

```
    C NUMBER;
```

```
BEGIN
```

```
    A:=100;
```

```
    B:=60;
```

```
    CALC_TAX_OUT(A,B,C);
```

```
END;
```

/

## Crear procedimiento con parámetros IN OUT

```
CREATE PROCEDURE CALC_TAX_IN_OUT (EMPL IN EMPLOYEES.EMPLOYEE_ID%TYPE,  
                                T1 IN OUT NUMBER)  
  
AS  
    SAL NUMBER:=0;  
    EXEC_USER EXCEPTION;  
  
BEGIN  
    IF T1<0 OR T1>60 THEN  
        RAISE EXEC_USER;  
    ELSE  
        SELECT SALARY INTO SAL FROM EMPLOYEES WHERE EMPLOYEE_ID=EMPL;  
        T1:=SAL*T1/100;  
    END IF;  
    DBMS_OUTPUT.PUT_LINE('SALARY: '||SAL);  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
        DBMS_OUTPUT.PUT_LINE('NO EXISTE EL EMPLEADO.');
```

WHEN EXEC\_USER THEN  
 DBMS\_OUTPUT.PUT\_LINE('TAX DEBE SER ENTRE 0 Y 60.');

```
END;  
  
/  
  
SET SERVEROUTPUT ON  
  
DECLARE  
    A NUMBER;  
    B NUMBER;  
  
BEGIN  
    A:=100;  
    B:=60;  
    CALC_TAX_IN_OUT(A,B);  
END;  
  
/
```

## Funciones

```
CREATE OR REPLACE FUNCTION CALC_TAX_F (EMPL IN EMPLOYEES.EMPLOYEE_ID%TYPE,
                                     T1 IN NUMBER)
RETURN NUMBER
AS
    TAX NUMBER:=0;
    SAL NUMBER:=0;
    EXEC_USER EXCEPTION;
BEGIN
    IF T1<0 OR T1>60 THEN
        RAISE EXEC_USER;
    ELSE
        SELECT SALARY INTO SAL FROM EMPLOYEES WHERE EMPLOYEE_ID=EMPL;
        TAX:=SAL*T1/100;
        DBMS_OUTPUT.PUT_LINE('SALARIO: '||SAL);
        RETURN TAX;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('NO EXISTE EL EMPLEADO.');
```

```
        RETURN TAX;
    WHEN EXEC_USER THEN
        DBMS_OUTPUT.PUT_LINE('TAX DEBE SER ENTRE 0 Y 60.');
```

```
        RETURN TAX;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('SUCESO INESPERADO');
```

```
END;
/

SET SERVEROUTPUT ON
DECLARE
    R NUMBER;
BEGIN
    R:=CALC_TAX_F(207,60);
    DBMS_OUTPUT.PUT_LINE('R= '||R);
END;
/
```

## Funciones con SQL

```
SELECT FIRST_NAME, LAST_NAME, CALC_TAX_F(EMPLOYEE_ID,10)FROM EMPLOYEES;
```

/

## Prácticas con procedimientos y funciones

### Práctica 1

```
CREATE PROCEDURE VISUALIZAR
AS
    NOMBRE VARCHAR2(100);
    SALARIO NUMBER;
BEGIN
    FOR EMPL IN (SELECT * FROM EMPLOYEES) LOOP
        NOMBRE:=EMPL.FIRST_NAME||' '||EMPL.LAST_NAME;
        SALARIO:=EMPL.SALARY;
        DBMS_OUTPUT.PUT_LINE(NOMBRE||' '||SALARIO);
    END LOOP;
END;
/
EXECUTE VISUALIZAR;
/
```

### Práctica 2

```
CREATE PROCEDURE VISUALIZAR_OUT (T1 IN NUMBER,
                                R1 OUT NUMBER)
AS
    EMPL EMPLOYEES%ROWTYPE;
    NOMBRE VARCHAR2(100);
    SALARIO NUMBER;
BEGIN
    R1:=0;
    FOR EMPL IN (SELECT * FROM EMPLOYEES WHERE DEPARTMENT_ID=T1) LOOP
        NOMBRE:=EMPL.FIRST_NAME||' '||EMPL.LAST_NAME;
        SALARIO:=EMPL.SALARY;
        DBMS_OUTPUT.PUT_LINE(NOMBRE||' '||SALARIO);
        R1:=R1+1;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE(R1);
END;
/

SET SERVEROUTPUT ON
```

```

DECLARE
  A NUMBER;
  B NUMBER;
BEGIN
  A:=80;
  VISUALIZAR_OUT(A,B);
END;

CREATE PROCEDURE FORMATO_CUENTA (NUMERO IN OUT VARCHAR2)
AS
  guardar1 VARCHAR2(20);
  guardar2 VARCHAR2(20);
  guardar3 VARCHAR2(20);
  guardar4 VARCHAR2(20);
BEGIN
  guardar1:=substr(numero,1,4);
  guardar2:=substr(numero,5,4);
  guardar3:=substr(numero,9,2);
  guardar4:=substr(numero,10);
  numero:=guardar1 || '-' || guardar2 || '-' || guardar3 || '-' || guardar4;
END;
/

DECLARE
  x varchar2(30):='15210457901111111111';
BEGIN
  FORMATO_CUENTA(x);
  dbms_output.put_line(x);
END;
/

```

## Práctica 3

```
CREATE OR REPLACE FUNCTION TOTAL_SALARY (CODIGO IN EMPLOYEES.DEPARTMENT_ID%TYPE)
RETURN NUMBER
AS
    DEPART NUMBER;
    TOTAL_SALARY NUMBER;
    TOTAL_EMPL NUMBER;
    USER_EXEC EXCEPTION;
BEGIN
    TOTAL_SALARY:=0;
    TOTAL_EMPL:=0;
    SELECT DEPARTMENT_ID INTO DEPART FROM DEPARTMENTS WHERE DEPARTMENT_ID=CODIGO;
    FOR EMPL IN (SELECT * FROM EMPLOYEES WHERE DEPARTMENT_ID=DEPART)LOOP
        TOTAL_SALARY:=TOTAL_SALARY+EMPL.SALARY;
        TOTAL_EMPL:=TOTAL_EMPL+1;
    END LOOP;

    IF TOTAL_EMPL = 0 THEN
        RAISE USER_EXEC;
    ELSE
        RETURN TOTAL_SALARY;
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('DEPARTAMENTO NO EXISTE');
        RETURN TOTAL_SALARY;
    WHEN USER_EXEC THEN
        DBMS_OUTPUT.PUT_LINE('NO EXISTEN EMPLEADOS');
        RETURN TOTAL_SALARY;
END;
/

SET SERVEROUTPUT ON
DECLARE
    CODIGO NUMBER;
    R1 NUMBER;
BEGIN
    CODIGO:=270;
    R1:=TOTAL_SALARY(CODIGO);
    DBMS_OUTPUT.PUT_LINE('TOTAL SALARY: '||R1);
END;
/
```



## Práctica 4

```
CREATE OR REPLACE FUNCTION TOTAL_SALARY_2 (CODIGO IN EMPLOYEES.DEPARTMENT_ID%TYPE,
      EMPL OUT NUMBER)
RETURN NUMBER
AS
  DEPART NUMBER;
  TOTAL_SALARY NUMBER;
  TOTAL_EMPL NUMBER;
  USER_EXEC EXCEPTION;
BEGIN
  TOTAL_SALARY:=0;
  TOTAL_EMPL:=0;
  SELECT DEPARTMENT_ID INTO DEPART FROM DEPARTMENTS WHERE DEPARTMENT_ID=CODIGO;
  FOR EMPL IN (SELECT * FROM EMPLOYEES WHERE DEPARTMENT_ID=DEPART)LOOP
    TOTAL_SALARY:=TOTAL_SALARY+EMPL.SALARY;
    TOTAL_EMPL:=TOTAL_EMPL+1;
  END LOOP;
  EMPL:=TOTAL_EMPL;
  IF TOTAL_EMPL = 0 THEN
    RAISE USER_EXEC;
  ELSE
    RETURN TOTAL_SALARY;
  END IF;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('DEPARTAMENTO NO EXISTE');
    RETURN TOTAL_SALARY;
  WHEN USER_EXEC THEN
    DBMS_OUTPUT.PUT_LINE('NO EXISTEN EMPLEADOS');
    RETURN TOTAL_SALARY;
END;
/
SET SERVEROUTPUT ON
DECLARE
  CODIGO NUMBER;
  R1 NUMBER;
  B NUMBER;
BEGIN
  CODIGO:=50;
```

```
R1:=TOTAL_SALARY_2(CODIGO,B);  
DBMS_OUTPUT.PUT_LINE('TOTAL SALARY: '||R1);  
DBMS_OUTPUT.PUT_LINE('TOTAL EMP: '||B);  
END;  
/
```

## Práctica 5

```
CREATE OR REPLACE FUNCTION INSERT_REGION(NOMBRE VARCHAR2)
RETURN NUMBER
AS
    CODIGO NUMBER:=0;
    REG REGIONS%ROWTYPE;
BEGIN
    SELECT MAX(REGION_ID) INTO CODIGO FROM REGIONS;
    CODIGO:=CODIGO+1;

    REG.REGION_ID:=CODIGO;
    REG.REGION_NAME:=NOMBRE;
    INSERT INTO REGIONS VALUES REG;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('REGION INSERTADA EXITOSAMENTE');
    RETURN CODIGO;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('SUCESO INESPERADO');
        RETURN CODIGO;
END;
/
SET SERVEROUTPUT ON
DECLARE
    NOMBRE VARCHAR2(100);
    R1 NUMBER;
BEGIN
    NOMBRE:='NOMBRE';
    R1:=INSERT_REGION(NOMBRE);
    DBMS_OUTPUT.PUT_LINE(R1);
END;
/
```

# Paquetes

## Crear un paquete

```
CREATE OR REPLACE PACKAGE PK1
AS
    V1 NUMBER;
    V2 VARCHAR2(100);
END;
/
SET SERVEROUTPUT ON
BEGIN
    PK1.V1:=PK1.V1+10;
    DBMS_OUTPUT.PUT_LINE(PK1.V1);
END;
/
```

## Crear el cuerpo de un paquete

```
CREATE OR REPLACE PACKAGE PK1
AS
    PROCEDURE CONVERT (NAME VARCHAR2, CONVERSION_TYPE CHAR);
END;
/
CREATE PACKAGE BODY PK1
AS
    FUNCTION UP(NAME VARCHAR2)
    RETURN VARCHAR2
    AS
    BEGIN
        RETURN UPPER(NAME);
    END UP;
    FUNCTION DO(NAME VARCHAR2)
    RETURN VARCHAR2
    AS
    BEGIN
        RETURN LOWER(NAME);
    END DO;

    PROCEDURE CONVERT (NAME VARCHAR2, CONVERSION_TYPE CHAR)
    AS
    BEGIN
```

```
IF CONVERSION_TYPE='U' THEN
DBMS_OUTPUT.PUT_LINE(UP(NAME));
ELSIF CONVERSION_TYPE='L' THEN
DBMS_OUTPUT.PUT_LINE(DO(NAME));
ELSE
DBMS_OUTPUT.PUT_LINE('EL PARÁMETRO DEBE SER U ó L');
END IF;
END CONVERT;
END PK1;
/
SET SERVEROUTPUT ON
BEGIN
PK1.CONVERT('aaaa','U');
END;
/
```

## Funciones con paquetes

```
CREATE OR REPLACE PACKAGE PK2
AS
FUNCTION CONVERT (NAME VARCHAR2, CONVERSION_TYPE CHAR) RETURN VARCHAR2;
END;
/

CREATE PACKAGE BODY PK2
AS
FUNCTION UP(NAME VARCHAR2)
RETURN VARCHAR2
AS
BEGIN
    RETURN UPPER(NAME);
END UP;
FUNCTION DO(NAME VARCHAR2)
RETURN VARCHAR2
AS
BEGIN
    RETURN LOWER(NAME);
END DO;

FUNCTION CONVERT (NAME VARCHAR2, CONVERSION_TYPE CHAR) RETURN VARCHAR2
AS
BEGIN
    IF CONVERSION_TYPE='U' THEN
        RETURN(UP(NAME));
    ELSIF CONVERSION_TYPE='L' THEN
        RETURN(DO(NAME));
    ELSE
        DBMS_OUTPUT.PUT_LINE('EL PARÁMETRO DEBE SER U ó L');
    END IF;
END CONVERT;
END PK2;
/

SET SERVEROUTPUT ON
DECLARE
    V1 VARCHAR2(100);
BEGIN
    V1:=PK2.CONVERT('aaaa','U');
    DBMS_OUTPUT.PUT_LINE(V1);
END;
/
```

## Sobrecarga de procedimientos

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(PK3.COUNT_EMPLOYEES('Shipping'));
END;
/
```

## UTL\_FILE

set serveroutput on

create or replace PROCEDURE read\_file IS

string VARCHAR2(32767);

Vfile UTL\_FILE.FILE\_TYPE;

```
BEGIN
  -- Open FILE
  Vfile := UTL_FILE.FOPEN('EXERCISES','F1.txt','R');
  Loop
    begin
      --read line
      UTL_FILE.GET_LINE(Vfile,string);
      INSERT INTO F1 VALUES(string);
    EXCEPTION
      WHEN NO_DATA_FOUND THEN EXIT;
    end;
  end loop;
  -- close file
  UTL_FILE.FCLOSE(Vfile);
```

```
END;
```

```
/
```

```
BEGIN
  READ_FILE;
```

```
END;
```

```
/
```

# Triggers

## Crear un trigger

```
CREATE OR REPLACE TRIGGER INSERT_EMPLOYEE
AFTER INSERT ON REGIONS
BEGIN
    INSERT INTO LOG_TABLE VALUES('INSERCIÓN EN LA TABLA REGIONS', USER);
END;
/
SELECT * FROM LOG_TABLE;
/
INSERT INTO REGIONS VALUES (104,'NOMBRE');
/
```

## Before trigger

```
CREATE OR REPLACE TRIGGER TR1_REGIONS
BEFORE INSERT ON REGIONS
BEGIN
    IF USER<>'HR' THEN
        RAISE_APPLICATION_ERROR(-20000,'SOLO HR PUEDE INSERTAR EN REGIONS');
    END IF;
END;
/
```



## Trigger con eventos multiples

### Ejemplo 1

```
CREATE OR REPLACE TRIGGER TR1_REGIONS
BEFORE INSERT OR UPDATE OR DELETE
ON REGIONS
BEGIN
    IF USER<>'HR' THEN
        RAISE_APPLICATION_ERROR(-20000,'SOLO HR PUEDE TRABAJAR EN REGIONS');
    END IF;
END;
/
```

### Ejemplo 2

```
CREATE OR REPLACE TRIGGER TR1_REGIONS
BEFORE INSERT OR UPDATE OF REGION_NAME OR DELETE
ON REGIONS
BEGIN
    IF USER<>'HR' THEN
        RAISE_APPLICATION_ERROR(-20000,'SOLO HR PUEDE TRABAJAR EN REGIONS');
    END IF;
END;
/
```

## Controlar el evento

```
CREATE OR REPLACE TRIGGER TR1_REGIONS
BEFORE INSERT OR UPDATE OR DELETE
ON REGIONS
BEGIN
    IF INSERTING THEN
        INSERT INTO LOG_TABLE VALUES ('OEPRACION INSERT', USER);
    END IF;

    IF UPDATING THEN
        INSERT INTO LOG_TABLE VALUES ('OEPRACION UPDATE', USER);
    END IF;

    IF DELETING THEN
        INSERT INTO LOG_TABLE VALUES ('OEPRACION DELETE', USER);
    END IF;
END;
/
```

## Controlar el evento columnas

```
CREATE OR REPLACE TRIGGER TR1_REGIONS
BEFORE INSERT OR UPDATE OR DELETE
ON REGIONS
BEGIN
    IF INSERTING THEN
        INSERT INTO LOG_TABLE VALUES ('OEPRACION INSERT', USER);
    END IF;

    IF UPDATING ('REGION_NAME') THEN
        INSERT INTO LOG_TABLE VALUES ('OEPRACION UPDATE REGION_NAME', USER);
    END IF;

    IF UPDATING ('REGION_ID') THEN
        INSERT INTO LOG_TABLE VALUES ('OEPRACION UPDATE REGION_ID', USER);
    END IF;

    IF DELETING THEN
        INSERT INTO LOG_TABLE VALUES ('OEPRACION DELETE', USER);
    END IF;
END;
/
--ROW TRIGGER

COMMIT;
```

## Clausula WHEN

```
CREATE OR REPLACE TRIGGER TR1_REGIONS
BEFORE INSERT OR UPDATE OR DELETE
ON REGIONS
FOR EACH ROW
WHEN (NEW.REGION_ID>1000)
BEGIN
    IF INSERTING THEN
        :NEW.REGION_NAME:=UPPER(:NEW.REGION_NAME);
        INSERT INTO LOG_TABLE VALUES ('OEPRACION INSERT', USER);
    END IF;

    IF UPDATING ('REGION_NAME') THEN
        INSERT INTO LOG_TABLE VALUES ('OEPRACION UPDATE REGION_NAME', USER);
    END IF;

    IF UPDATING ('REGION_ID') THEN
        INSERT INTO LOG_TABLE VALUES ('OEPRACION UPDATE REGION_ID', USER);
    END IF;

    IF DELETING THEN
        INSERT INTO LOG_TABLE VALUES ('OEPRACION DELETE', USER);
    END IF;
END;
/
```

## Comprobar estado de los triggers

```
DESC USER_TRIGGERS;
SELECT TRIGGER_NAME, TRIGGER_TYPE, TRIGGERING_EVENT, ACTION_TYPE,TRIGGER_BODY
FROM USER_TRIGGERS;
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
FROM USER_OBJECTS
WHERE OBJECT_TYPE='TRIGGER';
/
```

## Triggers en modo comando

```
SELECT * FROM USER_ERRORS;

ALTER TRIGGER TR1_REGIONS COMPILE;

ALTER TRIGGER TR1_REGIONS DISABLE;

SELECT TRIGGER_NAME, STATUS

FROM USER_TRIGGERS;

ALTER TRIGGER TR1_REGIONS ENABLE;

/
```

## Trigger compuesto

```
CREATE OR REPLACE TRIGGER TR2_COMPOUND
FOR INSERT OR UPDATE OR DELETE
ON REGIONS
COMPOUND TRIGGER
  BEFORE STATEMENT IS BEGIN
    INSERT INTO LOG_TABLE VALUES('BEFORE STATEMENT',USER);
  END BEFORE STATEMENT;
  AFTER STATEMENT IS BEGIN
    INSERT INTO LOG_TABLE VALUES('AFTER STATEMENT',USER);
  END AFTER STATEMENT;
  BEFORE EACH ROW IS BEGIN
    INSERT INTO LOG_TABLE VALUES('BEFORE EACH ROW',USER);
  END BEFORE EACH ROW;
  AFTER EACH ROW IS BEGIN
    INSERT INTO LOG_TABLE VALUES('AFTER EACH ROW',USER);
  END AFTER EACH ROW;
END;

/

INSERT INTO REGIONS VALUES(1051,'NOMBRE');
COMMIT;

/
```

## Triggers tipo DDL

```
CREATE OR REPLACE TRIGGER TR3_DDL
BEFORE DROP ON HR.SCHEMA
BEGIN
    RAISE_APPLICATION_ERROR(-20000,'NO SE PUEDE BORRAR TABLAS');
END;
/
DROP TABLE LOG_TABLE;
/
```

# OBJECT

## Crear un objeto

```
CREATE OR REPLACE TYPE PRODUCTO AS OBJECT(  
  --ATRIBUTOS  
    CODIGO NUMBER,  
    NOMBRE VARCHAR2(100),  
    PRECIO NUMBER,  
  --MÉTODOS  
    MEMBER FUNCTION ver_producto RETURN VARCHAR2,  
    MEMBER FUNCTION ver_precio RETURN NUMBER,  
    MEMBER PROCEDURE cambiar_precio(pvp number)  
);  
/  
DROP TYPE PRODUCTO;  
/
```

## Crear BODY del objeto

```
CREATE OR REPLACE TYPE BODY PRODUCTO AS  
  MEMBER FUNCTION ver_producto RETURN VARCHAR2 AS  
  BEGIN  
    RETURN 'CODIGO--->'||CODIGO||' NOMBRE--->'||NOMBRE||' PRECIO--->'||PRECIO;  
  END;  
  
  MEMBER FUNCTION ver_precio RETURN NUMBER AS  
  BEGIN  
    RETURN PRECIO;  
  END;  
  
  MEMBER PROCEDURE cambiar_precio(pvp NUMBER)AS  
  BEGIN  
    PRECIO:=PVP;  
  END;  
END;  
/  
--PROBAR PRODUCTO  
SET SERVEROUTPUT ON  
DECLARE  
  V1 PRODUCTO;  
BEGIN
```

```

V1:=PRODUCTO(100,'MANZANAS',10);
DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO());
DBMS_OUTPUT.PUT_LINE(V1.VER_PRODUCTO());
V1.CAMBIAR_PRECIO(20);
DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO());
V1.NOMBRE:='PERA';
DBMS_OUTPUT.PUT_LINE(V1.VER_PRODUCTO());
END;
/

```

## SELF

```

CREATE OR REPLACE TYPE PRODUCTO AS OBJECT(
--ATRIBUTOS
    CODIGO NUMBER,
    NOMBRE VARCHAR2(100),
    PRECIO NUMBER,
--MÉTODOS
    MEMBER FUNCTION ver_producto RETURN VARCHAR2,
    MEMBER FUNCTION ver_precio RETURN NUMBER,
--SELF
    MEMBER PROCEDURE cambiar_precio(PRECIO number)
);
/

CREATE OR REPLACE TYPE BODY PRODUCTO AS
    MEMBER FUNCTION ver_producto RETURN VARCHAR2 AS
    BEGIN
    RETURN 'CODIGO--->'||CODIGO||' NOMBRE--->'||NOMBRE||' PRECIO--->'||PRECIO;
    END;

    MEMBER FUNCTION ver_precio RETURN NUMBER AS
    BEGIN
    RETURN PRECIO;
    END;

    MEMBER PROCEDURE cambiar_precio(PRECIO NUMBER)AS
    BEGIN
    SELF.PRECIO:=PRECIO;
    END;
END;
/

```

```

SET SERVEROUTPUT ON
DECLARE
    V1 PRODUCTO;
BEGIN
    V1:=PRODUCTO(100,'MANZANAS',10);
    DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO());
    DBMS_OUTPUT.PUT_LINE(V1.VER_PRODUCTO());
    V1.CAMBIAR_PRECIO(20);
    DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO());
    V1.NOMBRE:='PERA';
    DBMS_OUTPUT.PUT_LINE(V1.VER_PRODUCTO());
END;
/

```

### Métodos estáticos (globales)

```

CREATE OR REPLACE TYPE PRODUCTO AS OBJECT(
--ATRIBUTOS
    CODIGO NUMBER,
    NOMBRE VARCHAR2(100),
    PRECIO NUMBER,
--MÉTODOS
    MEMBER FUNCTION ver_producto RETURN VARCHAR2,
    MEMBER FUNCTION ver_precio RETURN NUMBER,
--SELF
    MEMBER PROCEDURE cambiar_precio(PRECIO number),
--GLOBALES
    STATIC PROCEDURE AUDITORIA
);
/

CREATE OR REPLACE TYPE BODY PRODUCTO AS
    MEMBER FUNCTION ver_producto RETURN VARCHAR2 AS
    BEGIN
    RETURN 'CODIGO--->'||CODIGO||' NOMBRE--->'||NOMBRE||' PRECIO--->'||PRECIO;
    END;

    MEMBER FUNCTION ver_precio RETURN NUMBER AS
    BEGIN
    RETURN PRECIO;
    END;

```



```

MEMBER PROCEDURE cambiar_precio(PRECIO NUMBER)AS
BEGIN
SELF.PRECIO:=PRECIO;
END;

STATIC PROCEDURE AUDITORIA AS
BEGIN
INSERT INTO AUDITORIA VALUES(USER,SYSDATE);
END;
END;
/
SET SERVEROUTPUT ON
DECLARE
V1 PRODUCTO;
BEGIN
V1:=PRODUCTO(100,'MANZANAS',10);
DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO());
DBMS_OUTPUT.PUT_LINE(V1.VER_PRODUCTO());
V1.CAMBIAR_PRECIO(20);
DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO());
V1.NOMBRE:='PERA';
DBMS_OUTPUT.PUT_LINE(V1.VER_PRODUCTO());
PRODUCTO.AUDITORIA();
END;
/
CREATE TABLE AUDITORIA(
USUARIO VARCHAR2(100),
FECHA DATE);
/
SELECT * FROM AUDITORIA;
/
DECLARE
FECHA DATE;
BEGIN
FECHA:=SYSDATE();
INSERT INTO AUDITORIA VALUES('HR',FECHA);
END;
/
INSERT INTO AUDITORIA VALUES('HR','17-NOV-2024');
/

```

## Método constructor

```
CREATE OR REPLACE TYPE PRODUCTO AS OBJECT(  
  --ATRIBUTOS  
    CODIGO NUMBER,  
    NOMBRE VARCHAR2(100),  
    PRECIO NUMBER,  
  --MÉTODOS  
    MEMBER FUNCTION ver_producto RETURN VARCHAR2,  
    MEMBER FUNCTION ver_precio RETURN NUMBER,  
  --SELF  
    MEMBER PROCEDURE cambiar_precio(PRECIO number),  
  --GLOBALES  
    STATIC PROCEDURE AUDITORIA,  
  --MÉTODO CONSTRUCTOR  
    CONSTRUCTOR FUNCTION PRODUCTO (N1 VARCHAR2) RETURN SELF AS RESULT  
);  
/  
  
CREATE OR REPLACE TYPE BODY PRODUCTO AS  
  MEMBER FUNCTION ver_producto RETURN VARCHAR2 AS  
  BEGIN  
    RETURN 'CODIGO--->'||CODIGO||' NOMBRE--->'||NOMBRE||' PRECIO--->'||PRECIO;  
  END;  
  
  MEMBER FUNCTION ver_precio RETURN NUMBER AS  
  BEGIN  
    RETURN PRECIO;  
  END;  
  
  MEMBER PROCEDURE cambiar_precio(PRECIO NUMBER)AS  
  BEGIN  
    SELF.PRECIO:=PRECIO;  
  END;  
  
  STATIC PROCEDURE AUDITORIA AS  
  BEGIN  
    INSERT INTO AUDITORIA VALUES(USER,SYSDATE);  
  END;  
  
  CONSTRUCTOR FUNCTION PRODUCTO (N1 VARCHAR2) RETURN SELF AS RESULT  
  AS
```

```

BEGIN
SELF.NOMBRE:=N1;
SELF.PRECIO:=NULL;
SELF.CODIGO:=SEQ1.NEXTVAL;
RETURN;
END;
END;
/
DROP SEQUENCE SEQ1;
CREATE SEQUENCE SEQ1;
/
SET SERVEROUTPUT ON
DECLARE
    V1 PRODUCTO;
BEGIN
    V1:=PRODUCTO('MANZANAS');
    DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO());
    DBMS_OUTPUT.PUT_LINE(V1.VER_PRODUCTO());
    V1.CAMBIAR_PRECIO(20);
    DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO());
    V1.NOMBRE:='PERA';
    DBMS_OUTPUT.PUT_LINE(V1.VER_PRODUCTO());
    PRODUCTO.AUDITORIA();
END;
/

```

### Comprobar objetos

```

DESC PRODUCTO;
SELECT * FROM USER_TYPES;
SELECT TEXT FROM USER_SOURCE WHERE NAME='PRODUCTO';
/

```

### Sobrecarga de métodos

```

CREATE OR REPLACE TYPE PRODUCTO AS OBJECT(
--ATRIBUTOS
    CODIGO NUMBER,
    NOMBRE VARCHAR2(100),
    PRECIO NUMBER,
--MÉTODOS
    MEMBER FUNCTION ver_producto RETURN VARCHAR2,

```

```

    MEMBER FUNCTION ver_precio RETURN NUMBER,
    MEMBER FUNCTION ver_precio(impuestos NUMBER) RETURN NUMBER,
--SELF
    MEMBER PROCEDURE cambiar_precio(PRECIO number),
--GLOBALES
    STATIC PROCEDURE AUDITORIA,
--MÉTODO CONSTRUCTOR
    CONSTRUCTOR FUNCTION PRODUCTO (N1 VARCHAR2) RETURN SELF AS RESULT
);
/

CREATE OR REPLACE TYPE BODY PRODUCTO AS
    MEMBER FUNCTION ver_producto RETURN VARCHAR2 AS
    BEGIN
    RETURN 'CODIGO--->'||CODIGO||' NOMBRE--->'||NOMBRE||' PRECIO--->'||PRECIO;
    END;

    MEMBER FUNCTION ver_precio RETURN NUMBER AS
    BEGIN
    RETURN PRECIO;
    END;

    MEMBER FUNCTION ver_precio(impuestos NUMBER) RETURN NUMBER AS
    BEGIN
    RETURN PRECIO-PRECIO*IMPUESTOS/100;
    END;

    MEMBER PROCEDURE cambiar_precio(PRECIO NUMBER)AS
    BEGIN
    SELF.PRECIO:=PRECIO;
    END;

    STATIC PROCEDURE AUDITORIA AS
    BEGIN
    INSERT INTO AUDITORIA VALUES(USER,SYSDATE);
    END;

    CONSTRUCTOR FUNCTION PRODUCTO (N1 VARCHAR2) RETURN SELF AS RESULT
    AS
    BEGIN
    SELF.NOMBRE:=N1;

```

```

SELF.PRECIO:=NULL;
SELF.CODIGO:=SEQ1.NEXTVAL;
RETURN;
END;

END;
/
SET SERVEROUTPUT ON
DECLARE
    V1 PRODUCTO:= PRODUCTO(100,'MANZANAS',20);
BEGIN
    DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO());
    DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO(10));
END;
/

```

## Herencia

```

CREATE OR REPLACE TYPE PRODUCTO AS OBJECT(
--ATRIBUTOS
    CODIGO NUMBER,
    NOMBRE VARCHAR2(100),
    PRECIO NUMBER,
--MÉTODOS
    MEMBER FUNCTION ver_producto RETURN VARCHAR2,
    MEMBER FUNCTION ver_precio RETURN NUMBER,
    MEMBER FUNCTION ver_precio(impuestos NUMBER) RETURN NUMBER,
--SELF
    MEMBER PROCEDURE cambiar_precio(PRECIO number),
--GLOBALES
    STATIC PROCEDURE AUDITORIA,
--MÉTODO CONSTRUCTOR
    CONSTRUCTOR FUNCTION PRODUCTO (N1 VARCHAR2) RETURN SELF AS RESULT
)
--HERENCIA
    NOT FINAL
;
/
CREATE OR REPLACE TYPE COMESTIBLES UNDER PRODUCTO(
CADUCIDAD DATE,
MEMBER FUNCTION VER_CADUCIDAD RETURN VARCHAR2

```

```

);
/
CREATE OR REPLACE TYPE BODY COMESTIBLES AS
MEMBER FUNCTION VER_CADUCIDAD RETURN VARCHAR2 AS
BEGIN
RETURN CADUCIDAD;
END;
END;
/
SET SERVEROUTPUT ON
DECLARE
V1 COMESTIBLES:=COMESTIBLES(900,'TORNILLOS',20,SYSDATE());
BEGIN
DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO());
DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO(10));
DBMS_OUTPUT.PUT_LINE(V1.VER_CADUCIDAD);
END;
/

```

### Sobreescribir métodos

```

CREATE OR REPLACE TYPE COMESTIBLES UNDER PRODUCTO(
CADUCIDAD DATE,
MEMBER FUNCTION VER_CADUCIDAD RETURN VARCHAR2,
OVERRIDING MEMBER FUNCTION VER_PRECIO RETURN NUMBER
);
/
CREATE OR REPLACE TYPE BODY COMESTIBLES AS
MEMBER FUNCTION VER_CADUCIDAD RETURN VARCHAR2 AS
BEGIN
RETURN CADUCIDAD;
END;

OVERRIDING MEMBER FUNCTION VER_PRECIO RETURN NUMBER AS
BEGIN
RETURN PRECIO + 10;
END;
END;
/
SET SERVEROUTPUT ON
DECLARE

```

```

V1 COMESTIBLES := COMESTIBLES(900,'TORNILLOS',20,SYSDATE());
BEGIN
  DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO());
  DBMS_OUTPUT.PUT_LINE(V1.VER_PRECIO(10));
  DBMS_OUTPUT.PUT_LINE(V1.VER_CADUCIDAD());
END;
/

```

### Crear columna tipo objeto

```

CREATE TABLE TIENDA(
  CODIGO NUMBER,
  ESTANTERIA NUMBER,
  PRODUCTO PRODUCTO
);
/
INSERT INTO TIENDA VALUES(1, 1, PRODUCTO(1,'MANZANAS',20));
COMMIT;
/
SELECT * FROM TIENDA;
SELECT T.PRODUCTO.PRECIO FROM TIENDA T;
SELECT T.PRODUCTO.VER_PRODUCTO() FROM TIENDA T;
/

```

### Crear una tabla con campo constraint check JSON

```

CREATE TABLE PRODUCTOS(
  CODIGO INT,
  NOMBRE VARCHAR2(100),
  DATOS VARCHAR2(100)
  CONSTRAINT x1 CHECK (DATOS IS JSON)
);
/
INSERT INTO PRODUCTOS VALUES (
  1,
  'EJEMPLO 1',
  {
    "PAIS": "EGIPTO",
    "CIUDAD": "EL CAIRO",
    "POBLACION": 100
  }
);

```

```

}
');
/
SELECT * FROM PRODUCTOS;
/
--CREAR TABLA CON CAMPO JSON
CREATE TABLE PRODUCTOS1(
CODIGO INT,
NOMBRE VARCHAR2(100),
DATOS JSON
);
/
INSERT INTO PRODUCTOS1 VALUES (
1,
'EJEMPLO 1',
'
{
"PAIS": "EGIPTO",
"CIUDAD": "EL CAIRO",
"POBLACION": 100
}
');
COMMIT;
/
SELECT * FROM PRODUCTOS1;
/
--ACCEDER A DATOS DE TIPO JSON
INSERT INTO PRODUCTOS1 VALUES (
2,
'EJEMPLO 2',
'
{
"PAIS": "EGIPTO",
"CIUDAD": "EL CAIRO",
"POBLACION": 100,
"DIRECCION":{
    "CALLE": "JUAREZ",
    "PISO": "6",
    "PUERTA": "A"
  },

```



```

"TELEFONOS":[
    "1111-11111",
    "2222-22222"
]
}
');
COMMIT;
/
SELECT P1.DATOS.DIRECCION FROM PRODUCTOS1 P1;
SELECT P1.DATOS.DIRECCION.PISO FROM PRODUCTOS1 P1;
SELECT P1.DATOS.TELEFONOS FROM PRODUCTOS1 P1;
SELECT P1.DATOS.TELEFONOS[0] FROM PRODUCTOS1 P1;
SELECT P1.DATOS.TELEFONOS[1] FROM PRODUCTOS1 P1;
/

```

## Almacenar datos tipo CLOB

```

CREATE TABLE EJEMPLO11(
CODIGO INT,
FICHERO CLOB --ALMACENA DISTINTOS TIPOS DE DATOS
);
/
INSERT INTO EJEMPLO11 VALUES(1,{'"COLUMA1": "TEXTO_EJEMPLO"}');
INSERT INTO EJEMPLO11 VALUES(2,'EJEMPLO 2');
INSERT INTO EJEMPLO11 VALUES(3,'<doc><col1> prueba </col1></doc>');
COMMIT;
/
SELECT * FROM EJEMPLO11 WHERE FICHERO IS JSON;
SELECT * FROM EJEMPLO11 WHERE FICHERO IS NOT JSON;
/

```

## Objeto JSON

### JSON\_EXISTS

```
--JSON_EXISTS(campo_json,expresion_json,on_error);
--VERIFICA SI UN ATRIBUTO JSON EXISTE
SELECT DATOS FROM PRODUCTOS1;
/
INSERT INTO PRODUCTOS1 VALUES (
3,
'EJEMPLO 3',
',
{
"PAIS": "ANGOLA",
"CIUDAD": "",
"POBLACION":100,
"DIRECCION":{
    "CALLE": "JUAREZ",
    "PISO": "6",
    "PUERTA": "A"
},
"TELEFONOS":[
    "1111-11111",
    "2222-22222"
]
}
');
COMMIT;
/
SELECT PROD1.DATOS FROM PRODUCTOS1 PROD1 WHERE
JSON_EXISTS(PROD1.DATOS,'$.DIRECCION');
/
```

### JSON\_VALUE

```
--RECUPERA DATOS SIMPLES
SELECT JSON_VALUE(PROD1.DATOS,'$.PAIS') FROM PRODUCTOS1 PROD1;
SELECT JSON_VALUE(PROD1.DATOS,'$.DIRECCION') FROM PRODUCTOS1 PROD1;
SELECT JSON_VALUE(PROD1.DATOS,'$.DIRECCION.CALLE') FROM PRODUCTOS1 PROD1;
/
```

## JSON\_QUERY

--RECUPERA DATOS COMPLEJOS

```
SELECT JSON_QUERY(PROD1.DATOS,'$.PAIS') FROM PRODUCTOS1 PROD1;
SELECT JSON_QUERY(PROD1.DATOS,'$.DIRECCION') FROM PRODUCTOS1 PROD1;
SELECT JSON_QUERY(PROD1.DATOS,'$.DIRECCION.PISO') FROM PRODUCTOS1 PROD1;
/
```

## JSON\_TABLE

```
SELECT J_PAIS FROM PRODUCTOS1 PROD1, JSON_TABLE(PROD1.DATOS,'$' COLUMNS(J_PAIS PATH
'$.PAIS'));
SELECT J_PAIS, J_CIUADAD FROM PRODUCTOS1 PROD1, JSON_TABLE(PROD1.DATOS,'$' COLUMNS
(J_PAIS PATH '$.PAIS',
J_CIUADAD PATH '$.CIUDAD'));
SELECT J_PAIS, J_CIUADAD, J_DIRECCION FROM PRODUCTOS1 PROD1, JSON_TABLE(PROD1.DATOS,'$'
COLUMNS
(J_PAIS PATH '$.PAIS',
J_CIUADAD PATH '$.CIUDAD',
J_DIRECCION PATH '$.DIRECCION.CALLE'));
CREATE VIEW DATOS_DIRECCION AS SELECT J_PAIS, J_CIUADAD, J_DIRECCION FROM PRODUCTOS1
PROD1, JSON_TABLE(PROD1.DATOS,'$' COLUMNS
(J_PAIS PATH '$.PAIS',
J_CIUADAD PATH '$.CIUDAD',
J_DIRECCION PATH '$.DIRECCION.CALLE'));
SELECT * FROM DATOS_DIRECCION;
/
```

## JSON\_MERGEPATCH

```
SELECT DATOS FROM PRODUCTOS1;
/
UPDATE PRODUCTOS1 SET DATOS=
'{
"PAIS": "EGIPTO",
"CIUDAD": "EL CAIRO",
"POBLACION": 200
}' WHERE CODIGO=1;
/
UPDATE PRODUCTOS1 SET DATOS=
'{
"PAIS": "EGIPTO",
"CIUDAD": "EL CAIRO",
```

```

"POBLACION": 200,
"ESTADO": TRUE
}' WHERE CODIGO=1;
/
UPDATE PRODUCTOS1 SET DATOS=JSON_MERGEPATCH(DATOS,
'{'
"ESTADO": false
}'
)WHERE CODIGO=1;
/

```

## JSON\_TRANSFORM

--APARTIR DE LA 21C

### OPERACIONES

SET ----> ACTUALIZAR UN ELEMENTO PERO SINO EXISTE LO CREA

INSERT ----> INSERTA UN ELEMENTO

APPEND ----> AÑADIR UN ELEMENTO A UN ARRAY

REMOVE ----> BORRA UN ELEMENTO

RENAME ----> RENOMBRA UN ELEMENTO

REPLACE ----> ACTUALIZAR UN ELMENTO PERO SINO EXISTE NO LO CREA

KEEP ----> ELIMINA TODOS LOS ELEMENTOS SALVO LOS QUE ESTAN SELECCIONADOS

```

SELECT DATOS FROM PRODUCTOS1;
UPDATE PRODUCTOS1 SET DATOS=JSON_TRANSFORM(DATOS,
      SET '$.POBLACION'=500)
      WHERE CODIGO=1;
SELECT JSON_TRANSFORM(DATOS, SET '$.POBLACION'=1000)FROM PRODUCTOS1 WHERE CODIGO=1;
SELECT JSON_TRANSFORM(DATOS, INSERT '$.TIPO'='TIPO UNO')FROM PRODUCTOS1 WHERE
CODIGO=1;
SELECT JSON_TRANSFORM(DATOS, APPEND '$.TELEFONOS'='3333-333333')FROM PRODUCTOS1
WHERE CODIGO=2;
SELECT JSON_TRANSFORM(DATOS, RENAME '$.POBLACION'='POB')FROM PRODUCTOS1 WHERE
CODIGO=1;
SELECT JSON_TRANSFORM(DATOS, REPLACE '$.POBLACION1'='500')FROM PRODUCTOS1 WHERE
CODIGO=1;
SELECT JSON_TRANSFORM(DATOS, REMOVE '$.POBLACION')FROM PRODUCTOS1 WHERE CODIGO=1;

```

```

SELECT JSON_TRANSFORM(DATOS, KEEP '$.DIRECCION.CALLE')FROM PRODUCTOS1 WHERE
CODIGO=3;
SELECT JSON_TRANSFORM(DATOS, SET '$.POBLACION'=200,
        INSERT '$.DIRECCION.CODIGO'=90901,
        RENAME '$.DIRECCION.CALLE'='VIA')FROM PRODUCTOS1 WHERE CODIGO=3;
/

```

## JSON CON PL/SQL

```

SET SERVEROUTPUT ON
DECLARE
    C1 VARCHAR2(200);
BEGIN
    SELECT PROD1.DATOS.POBLACION INTO C1 FROM PRODUCTOS1 PROD1 WHERE CODIGO=3;
    DBMS_OUTPUT.PUT_LINE(C1);
    SELECT JSON_VALUE(PROD1.DATOS,'$.PAIS')INTO C1 FROM PRODUCTOS1 PROD1 WHERE
CODIGO=3;
    DBMS_OUTPUT.PUT_LINE(C1);
    SELECT JSON_VALUE(PROD1.DATOS,'$.DIRECCION')INTO C1 FROM PRODUCTOS1 PROD1 WHERE
CODIGO=3;
    DBMS_OUTPUT.PUT_LINE(C1);
    SELECT JSON_VALUE(PROD1.DATOS,'$.DIRECCION.CALLE')INTO C1 FROM PRODUCTOS1 PROD1
WHERE CODIGO=3;
    DBMS_OUTPUT.PUT_LINE(C1);
    SELECT JSON_QUERY(PROD1.DATOS,'$.DIRECCION')INTO C1 FROM PRODUCTOS1 PROD1 WHERE
CODIGO=3;
    DBMS_OUTPUT.PUT_LINE(C1);
    SELECT JSON_QUERY(PROD1.DATOS,'$.DIRECCION.CALLE')INTO C1 FROM PRODUCTOS1 PROD1
WHERE CODIGO=3;
    DBMS_OUTPUT.PUT_LINE(C1);
    SELECT JSON_TRANSFORM(DATOS, RENAME '$.POBLACION'='POB')INTO C1 FROM PRODUCTOS1
WHERE CODIGO=3;
    DBMS_OUTPUT.PUT_LINE(C1);
END;
/

```

## Comando PUT en JSON con PL/SQL

```
SET SERVEROUTPUT ON
DECLARE
  JSON1 JSON_OBJECT_T;
  V1 VARCHAR2(200);
BEGIN
  --CONSTRUCTOR
  JSON1:=JSON_OBJECT_T.PARSE('{ "NOMBRE": "ALBERTO"}');
  DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
  JSON1.PUT('EDAD',29);
  JSON1.PUT('TELEFONO','999999');
  DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
  --DOCUMENTO ANIDADO
  JSON1.PUT('DIRECCION',JSON_OBJECT_T('{ "CALLE": "PEZ", "NUMERO": 10,"CIUDAD": "AFRICA"}'));
  DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
  --ARRAY
  JSON1.PUT('EXPERIENCIA',JSON_ARRAY_T(['WORD','EXCEL','LINUX']));
  DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
  --ACTUALIZACION
  JSON1.PUT('EDAD',38);
  DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
  --RENOMBRAR PARAMETRO
  JSON1.RENAME_KEY('NOMBRE','NOM');
  DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
  --BORRAR UN ELEMENTO
  JSON1.REMOVE('TELEFONO');
  DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
  --SERIALIZACION SIRVE PARA RECUPERAR INFORMACIÓN
  V1:=JSON1.GET_STRING('NOM');
  DBMS_OUTPUT.PUT_LINE(V1);
  V1:=JSON1.GET_NUMBER('EDAD');
  DBMS_OUTPUT.PUT_LINE(V1);
  V1:=JSON1.GET_OBJECT('DIRECCION').GET_STRING('CALLE');
  DBMS_OUTPUT.PUT_LINE(V1);
END;
/
```

## Trabajar con la base de datos

```
SET SERVEROUTPUT ON
DECLARE
    JSON1 JSON_OBJECT_T;
    V1 VARCHAR2(200);
    RESULTADO VARCHAR2(200);
BEGIN
    SELECT DATOS INTO V1 FROM PRODUCTOS1 WHERE CODIGO=3;
    DBMS_OUTPUT.PUT_LINE(V1);
    JSON1:=JSON_OBJECT_T.PARSE(V1);
    DBMS_OUTPUT.PUT_LINE(JSON1.to_string);
    JSON1.PUT('COL','NUEVO_VALOR');
    DBMS_OUTPUT.PUT_LINE(JSON1.to_string);
    RESULTADO:=JSON1.TO_STRING;
    UPDATE PRODUCTOS1 SET DATOS=RESULTADO WHERE CODIGO=3;
    SELECT DATOS INTO V1 FROM PRODUCTOS1 WHERE CODIGO=3;
    DBMS_OUTPUT.PUT_LINE(V1);
END;
/
```

## Crear un ARRAY JSON

```
SET SERVEROUTPUT ON
DECLARE
    JSON1 JSON_ARRAY_T;
    V1 VARCHAR2(200);
BEGIN
    JSON1:=JSON_ARRAY_T(['UNO','DOS','TRES']);
    DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
    DBMS_OUTPUT.PUT_LINE(JSON1.GET_SIZE);
    --RECUPERAR VALOR DE UN ARRAY
    DBMS_OUTPUT.PUT_LINE(JSON1.GET(0).TO_STRING);
    --RECUPERAR TODOS LOS ELEMENTOS
    FOR X IN 0..JSON1.GET_SIZE-1 LOOP
        DBMS_OUTPUT.PUT_LINE(JSON1.GET(X).TO_STRING);
    END LOOP;
END;
/
```

## Documentos de un ARRAY

```
SET SERVEROUTPUT ON
DECLARE
  JSON1 JSON_ARRAY_T;
  V1 VARCHAR2(200);
BEGIN
  JSON1:=JSON_ARRAY_T(['{"CIUDAD":"MEXICO",
    "ARCHIVOS": ["UNO","DOS","TRES"]},
    {"CIUDAD":"EL SALVADOR",
    "ARCHIVOS": ["UNO","DOS","TRES"]}]);
  DBMS_OUTPUT.PUT_LINE(JSON1.GET_SIZE);
  DBMS_OUTPUT.PUT_LINE(JSON1.GET(0).TO_STRING);
  FOR X IN 0..JSON1.GET_SIZE-1 LOOP
    DBMS_OUTPUT.PUT_LINE(JSON1.GET(X).TO_STRING);
  END LOOP;
END;
/
```



## Funciones del ARRAY

```
SET SERVEROUTPUT ON
DECLARE
    JSON1 JSON_ARRAY_T;
    V1 VARCHAR2(200);
BEGIN
    JSON1:=JSON_ARRAY_T(['UNO','DOS','TRES']);
    DBMS_OUTPUT.PUT_LINE(JSON1.GET_SIZE);
    DBMS_OUTPUT.PUT_LINE(JSON1.GET(0).TO_STRING);
    FOR X IN 0..JSON1.GET_SIZE-1 LOOP
        DBMS_OUTPUT.PUT_LINE(JSON1.GET(X).TO_STRING);
    END LOOP;
    --AÑADIR UN ELEMENTO
    JSON1.PUT(2,'CUATRO');
    DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
    --AÑADIR UN NULL
    JSON1.APPEND_NULL();
    DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
    --AÑADIR UN ELEMENTO FINAL
    JSON1.APPEND('CINCO');
    DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
    --ELIMINAR ELEMENTO
    JSON1.REMOVE(3);
    DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
    --AÑADIR UN ARRAY
    JSON1.PUT(3,JSON_ARRAY_T(['V1','V2','V3']));
    DBMS_OUTPUT.PUT_LINE(JSON1.TO_STRING);
END;
/
```