

# PROGRAMACIÓN EN ORACLE



The screenshot shows the MySQL Query Browser interface. The main window displays a SQL script for creating tables in a MySQL database. The script is as follows:
 

```

1 -- *****
2 -- Copyright 1998-2008 Anywhere Solutions, Inc. All rights reserved.
3 -- *****
4
5
6 -- Create the Mobilink Server system tables and stored procedures in
7 -- a MySQL consolidated database.
8 --
9
10 delimiter //
11
12 create table m_user (
13   user_id integer not null auto increment,
14   name varchar( 128 ) not null unique,
15   hashed_password binary( 32 ) null,
16   primary key( user_id ) )
17 //
18
19
20 create table m_database (
21   rid integer not null auto increment,
22   remote_id varchar( 128 ) not null unique,
23   script_id datetime default '1900-01-01 00:00:00.000000' not null,
24   description varchar( 128 ) null,
25   primary key( rid ) )
26 //
27
28 create table m_subscription (
29   rid integer not null,
30   subscription_id integer not null,
31   user_id integer not null,
32   progress numeric( 20 ) default 0 not null,
33   publication_name varchar( 128 ) default '<unknown>' not null,
34   last_upload_time datetime default '1900-01-01 00:00:00' not null,
35   last_download_time datetime default '1900-01-01 00:00:00' not null,
36   primary key( rid, subscription_id ),
37   foreign key( rid ) references m_database ( rid ),
38   foreign key( user_id ) references m_user ( user_id ) )
39 //
40
41
42
    
```

 The right-hand side of the interface shows a 'Schemata' pane with a tree view containing 'information\_schema', 'mysql', and 'sys'. The 'mysql' database is selected and highlighted with a red circle. Below the schemata pane is a 'Syntax' pane with tabs for 'Functions', 'Params', and 'Tx'.

# Introducción

- SQL es un lenguaje utilizado en el ámbito de los sistemas de bases de datos relacionales.
- Uno de sus inconvenientes es que no posee la potencia de los lenguajes de programación. Esto implica que, cuando se desea realizar una aplicación para el manejo de una base de datos relacional, hay que acudir a una herramienta *externa que soporte la* capacidad operativa del SQL y la versatilidad de los lenguajes de programación tradicionales.
- **PL/SQL** (*Procedural Language/SQL*) es el lenguaje de programación que proporciona Oracle para extender la definición de SQL con instrucciones de procesamiento

# Sintaxis

- En PL/SQL los **comentarios** pueden ser:
  - De una sola línea, empezando por --.
  - Multilínea, entre los símbolos /\* y \*/.
- Para mostrar por el terminal cadenas o valores se debe ejecutar la instrucción:

**DBMS\_OUTPUT.PUT\_LINE(*dato*);**

- En el entorno de SQL\*Plus puede ser necesario habilitar la visualización ejecutando, previamente, la instrucción:

**SET SERVEROUTPUT ON**

# Bloques

- Los bloques PL/SQL presentan una estructura específica compuesta de tres secciones diferenciadas, delimitadas por las siguientes palabras reservadas:
  - DECLARE
  - **BEGIN**
  - EXCEPTION
  - **END;**
- En la sección de declaración se declaran todas las constantes y variables que se van a utilizar en la ejecución del bloque.
- La sección de ejecución incluye las instrucciones a ejecutar en el bloque PL/SQL. Es la única sección obligatoria.
- En la sección de excepciones se definen los manejadores de errores que soportará el bloque PL/SQL.

# Ejemplo bloque anónimo

- Ejemplo 1. Bloque PL/SQL que muestra un mensaje por pantalla:

```
BEGIN  
DBMS_OUTPUT.PUT_LINE('Hola mundo!!');  
END;
```

```
SQL>  
SQL> begin  
2  dbms_output.put_line('Hola!!!!');  
3  end;  
4  /  
Hola!!!!  
  
PL/SQL procedure successfully completed.  
SQL> _
```

# Declaración de variables

- La sección de declaración de variables de un bloque PL/SQL tiene la siguiente sintaxis:

```
nombre [CONSTANT] tipo [NOT NULL] [:=valor_inicial];
```

Tipo puede ser:

- Un tipo de dato simple: DATE, CHAR, NUMBER, BOOLEAN...
- El tipo de un atributo: *nombre%TYPE*.
- El tipo correspondiente a una tupla: *nombre%ROWTYPE*.

La cláusula CONSTANT indica que el valor\_inicial asignado no se puede modificar en el código.

La cláusula NOT NULL impide que se asigne valor nulo a la variable.

El valor inicial debe corresponder con el tipo definido. Si no se incluye, la variable toma el valor inicial NULL.

# Declaración de variables

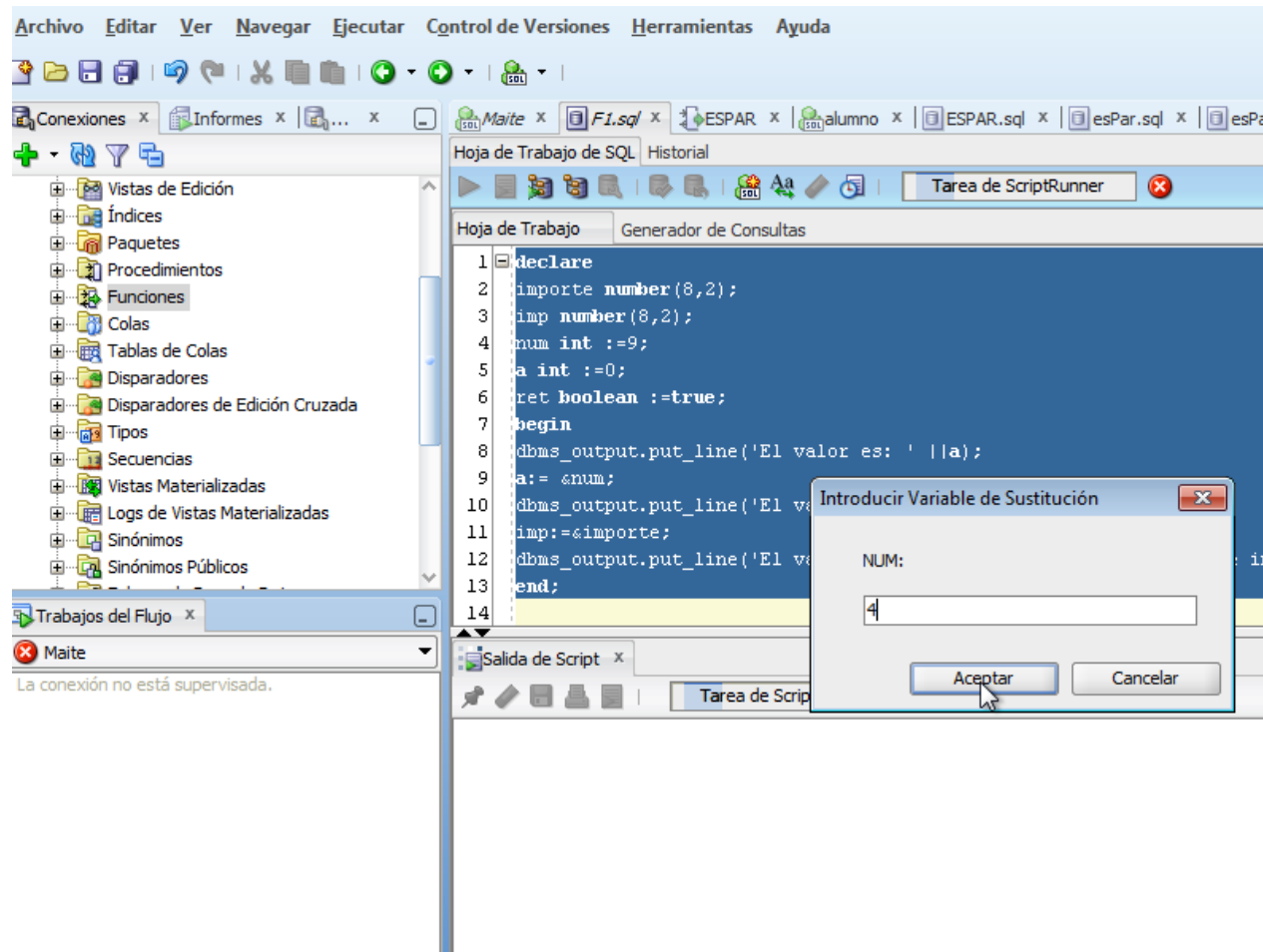
```
DECLARE
    precio NUMBER(5,2);
    localidad VARCHAR2(20) := 'Madrid';
    IVA CONSTANT NUMBER(2) := 18;
    apellidos empleado.apellidos%TYPE;
BEGIN
```

Ejemplo de asignación simple:

localidad:= 'Segovia';

Precio:=125.25;

# Codificar bloque anónimos





# Ejemplo1 bloques anónimos

```
set serveroutput on;
declare
base INT :=&NUMERO1;
altura NUMBER :=&NUMERO2;
area number(4,2);
begin
area:=base*altura/2;
dbms_output.put_line('El area es ' || area);
end;
/
```

# Ejemplo2 bloques anónimos

```
DECLARE
numero NUMBER:=&dameNumero;
CUADRADO NUMBER;
BEGIN
CUADRADO:=numero*numero;
DBMS_OUTPUT.PUT_LINE('EL NUMERO ' || numero
    || ' AL CUADRADO ES:' || CUADRADO);
END;
/
```

# Ejemplo3 bloques anónimos

```
/* Crear un programa script_IVA que pida un cantidad de dinero y  
muestre el resultado de aplicarle el 21 % de IVA. */
```

```
DECLARE
```

```
NUM INT :=0;
```

```
IVA CONSTANT REAL:=0.21;
```

```
RES REAL;
```

```
BEGIN
```

```
NUM:=&DAMEPrecio;
```

```
RES:= NUM+num*IVA;
```

```
dbms_output.put_line('El resultado es '||RES);
```

```
end;
```

```
/
```

# Instrucciones PL/SQL

- ☐ De asignación.
- ☐ De control de flujo.
- ☐ Bucles

- La instrucción básica de **asignación** es:

```
variable := expresion;
```

- Una expresión en PL/SQL puede ser:
  - ✓ Un literal.
  - ✓ Una variable o constante definida en el bloque.
  - ✓ El resultado de la ejecución de una función, cuyos argumentos pueden ser, a su vez, literales, constantes o variables.
  - ✓ Una combinación de los anteriores, unidos mediante operadores.

# Instrucciones de asignación

- PL/SQL proporciona un amplio catálogo de funciones, tanto de SQL como propias:
  - **De manejo de cadenas de caracteres:** INITCAP(), INSTR(), LENGTH(), LOWER(), LPAD(), LTRIM(), REPLACE(), RPAD(), RTRIM(), SUBSTR(), UPPER().
  - **Numéricas:** ABS(), CEIL(), FLOOR(), MOD(), POWER(), ROUND(), SIGN(), SQRT(), TRUNC().
  - **De manejo de fechas:** ADD\_MONTHS(), LAST\_DAY(), MONTHS\_BETWEEN(), NEXT\_DAY(), SYSDATE(), etc.
  - **De conversión:** TO\_CHAR(), TO\_DATE(), TO\_NUMBER().
  - **De control de errores:** SQLCODE(), SQLERRM().
  - **Otras:** UID(), USER(), DECODE(), etc

# Instrucciones de asignación

- En PL/SQL, se pueden utilizar operadores sobre:
  - **Números:** +, -, \*, /, \*\* (potencia), MOD (módulo).
  - **Cadenas:** || (concatenación).
  - **Lógicos:** AND, OR, NOT.
- Los valores lógicos aparecen como resultado de alguna comparación o verificación de valor.

En PL/SQL se pueden utilizar:

- Comparadores matemáticos: =, !=, <, <=, >, >=.
- Comparadores propios del lenguaje SQL: [NOT] LIKE, IS [NOT] NULL, [NOT] BETWEEN ... AND ..., [NOT] IN.

# Operadores lógicos

A	B	A AND B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	NULL	NULL
NULL	TRUE	NULL
FALSE	NULL	FALSE
NULL	FALSE	FALSE
FALSE	FALSE	FALSE

A	B	A OR B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
TRUE	NULL	TRUE
NULL	TRUE	TRUE
FALSE	NULL	NULL
NULL	FALSE	NULL
FALSE	FALSE	FALSE

A	NOT A
TRUE	FALSE
FALSE	TRUE
NULL	NULL

# Ejercicios Operadores y Expresiones

- Analizar las siguientes expresiones, indicando si su resultado sería TRUE, FALSE o NULL

a)  $(16-8)/2=4$  **V**

h) `'a12'='a' || '12'` **V**

b)  $(7+3) \neq 10$  **F**

i) `0 IS NULL` **F**

c) `0 IS NOT NULL` **V**

j) `'ANa' LIKE '%N'` **F**

d)  $(3=(9/3))$  and null **NULL**

k) `'B' IN ('A','D')` **F**

e) null or  $(2*5=10)$  **V**

f) 4 between 3 and 9 **V**

g) NOT  $(2**3=8)$  **F**



# Instrucciones de asignación

- Además de las instrucciones de asignación *clásicas*, *PL/SQL permite asignar el resultado de* la ejecución de una sentencia SQL a:
  - Una lista de variables, cuando el conjunto resultado devuelve **una única tupla (%TYPE)**.

```
nombreVariable nombreObjeto%TYPE;
```

- Un cursor, si no se conoce de antemano el número de filas que se va a generar como resultado, o es previsible que devuelva varias(**%ROWTYPE**)

```
nombreVariable nombreObjeto%ROWTYPE;
```

# Instrucciones de asignación

- Para asignar el resultado de una sentencia SELECT a una lista de variables se utiliza la sintaxis:

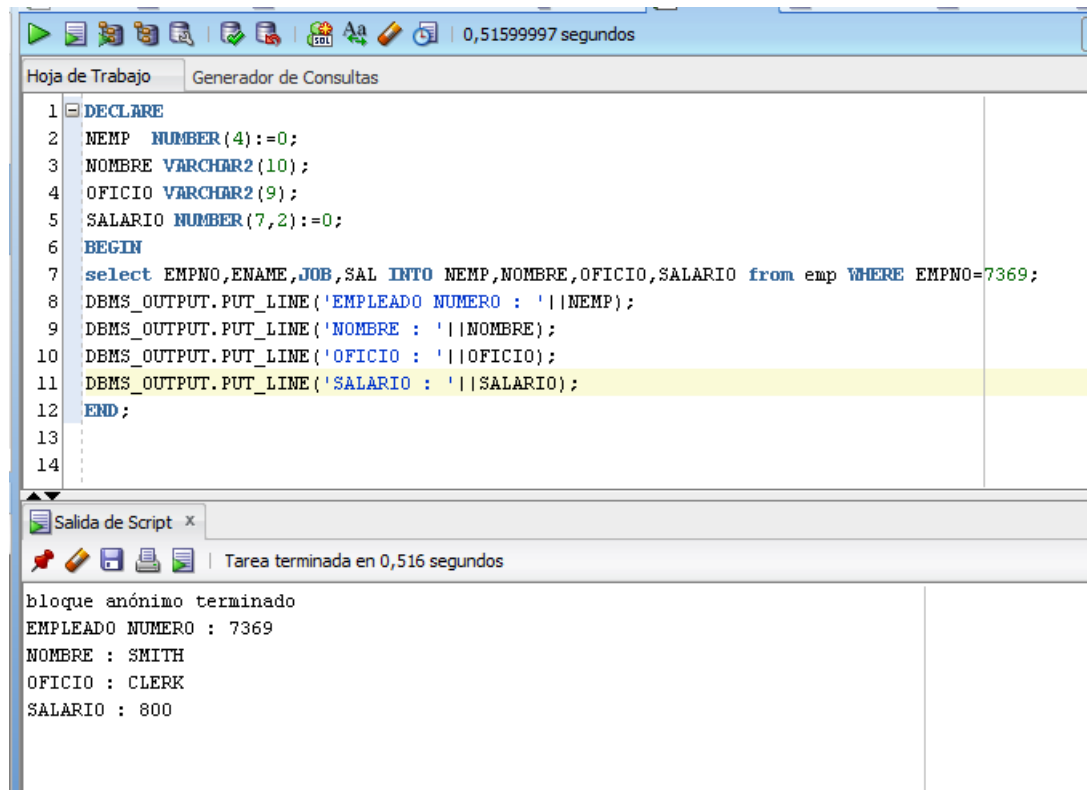
```
SELECT atributos INTO variables FROM...;
```

- El número y tipo de variables escalares en la lista de variables debe corresponderse con el número y tipo de atributos de la consulta.
- La lista de variables puede sustituirse por una variable *tupla del tipo correspondiente a la tabla*, si la consulta devuelve todos los atributos.

# Instrucciones de asignación

```
set serveroutput on;
DECLARE
fecha DATE;
BEGIN
-- La sentencia devuelve un único valor
SELECT SYSDATE INTO fecha FROM DUAL;
DBMS_OUTPUT.PUT_LINE('Hoy es '||TO_CHAR(fecha,'Day')||'
    '||to_char(fecha,'dd')||' de '|| TO_CHAR(fecha,'Mon')||'
    de '|| TO_CHAR(fecha,'yyYY'));
END;
/
```

# Instrucciones de asignación



```
1 DECLARE
2   NEMP  NUMBER(4) := 0;
3   NOMBRE VARCHAR2(10);
4   OFICIO VARCHAR2(9);
5   SALARIO NUMBER(7,2) := 0;
6 BEGIN
7   select EMPNO,ENAME,JOB,SAL INTO NEMP,NOMBRE,OFICIO,SALARIO from emp WHERE EMPNO=7369;
8   DBMS_OUTPUT.PUT_LINE('EMPLEADO NUMERO : '||NEMP);
9   DBMS_OUTPUT.PUT_LINE('NOMBRE : '||NOMBRE);
10  DBMS_OUTPUT.PUT_LINE('OFICIO : '||OFICIO);
11  DBMS_OUTPUT.PUT_LINE('SALARIO : '||SALARIO);
12 END;
```

Salida de Script x

Tarea terminada en 0,516 segundos

bloque anónimo terminado  
EMPLEADO NUMERO : 7369  
NOMBRE : SMITH  
OFICIO : CLERK  
SALARIO : 800

# Instrucciones de asignación

## **DECLARE**

```
VNEMP emple.emp_no%type;  
VNOMBRE EMPLE.nombre%TYPE;  
VOFICIO emple.oficio%TYPE;  
VSALARIO emple.salario%TYPE;
```

## **BEGIN**

```
select EMP_NO,nombre,oficio,SALario INTO  
    VNEMP,VNOMBRE,VOFICIO,VSALARIO from emple WHERE nombre  
    LIKE UPPER('Martin');  
DBMS_OUTPUT.PUT_LINE('EMPLEADO NUMERO : '||VNEMP);  
DBMS_OUTPUT.PUT_LINE('NOMBRE : '||VNOMBRE);  
DBMS_OUTPUT.PUT_LINE('OFICIO : '||VOFICIO);  
DBMS_OUTPUT.PUT_LINE('SALARIO : '||VSALARIO);  
END;
```

# Instrucciones de asignación

**DECLARE**

empleado emple%rowtype;

**BEGIN**

Select \* INTO empleado from emple WHERE NOMBRE LIKE  
UPPER('Martin');

DBMS\_OUTPUT.PUT\_LINE('EMPLEADO NUMERO :  
' || empleado.emp\_no);

DBMS\_OUTPUT.PUT\_LINE('NOMBRE : ' || empleado.nombre);

DBMS\_OUTPUT.PUT\_LINE('OFICIO : ' || empleado.oficio);

DBMS\_OUTPUT.PUT\_LINE('SALARIO : ' || empleado.salario);

**END;**

# ERROR SELECT INTO CON MULTIPLES REGISTROS

- Si existen varios registros que cumplan las condiciones, se producirá un error, ya que cada variable sólo puede almacenar un valor. Posteriormente veremos los CURSORES que permiten reutilizar las variables si hay más de un registro de resultado del SELECT.
- Podemos controlar cuantos registros se van a leer de dos formas:
  - 1) Averiguando con count(\*) el número de respuestas y almacenándola en una variable.
  - 2) Usando la variable SQL%ROWCOUNT que devuelve el número de registros afectados tras la consulta

# Ejemplo de SELECT INTO que devuelve más de un Registro

```
/* Visualizar el apellido, número y sueldo TOTAL de los empleados que empiezan por S*/  
DECLARE  
NUM EMPL.EMP_NO%TYPE;  
SUELDO EMPL.SALARIO%TYPE;  
APEL EMPL.APELLIDO%TYPE;  
CUANTOS NUMBER;  
BEGIN  
SELECT COUNT(*) INTO CUANTOS FROM EMPL WHERE UPPER(APELLIDO) LIKE 'S%';  
DBMS_OUTPUT.PUT_LINE('HAY ' || CUANTOS || ' EMPLEADOS QUE EMPIEZAN POR S');  
IF CUANTOS = 1 THEN  
SELECT EMP_NO, SALARIO+NVL(COMISION,0) INTO NUM, SUELDO  
FROM EMPL WHERE UPPER(APELLIDO) LIKE 'S%';  
DBMS_OUTPUT.PUT_LINE('HAY REGISTROS ' || SQL%ROWCOUNT);  
DBMS_OUTPUT.PUT_LINE('EL NUMERO DE EMPLEADO DEL QUE EMPIEZA POR S ES ' || NUM);  
DBMS_OUTPUT.PUT_LINE('EL SUELDO DEL EMPLEADO ES ' || SUELDO);  
ELSE  
DBMS_OUTPUT.PUT_LINE('HAY MAS DE UN EMPLEADO !!');  
END IF;  
END;  
/
```



# Ejemplo de SELECT INTO que devuelve más de un Registro con EXCEPTION

```
/* Visualizar el apellido, número y sueldo TOTAL de los empleados que empiezan por L*/  
DECLARE  
NUM EMPL.EMP_NO%TYPE;  
SUELDO EMPL.SALARIO%TYPE;  
APEL EMPL.APELLIDO%TYPE;  
CUANTOS NUMBER;  
BEGIN  
SELECT COUNT(*) INTO CUANTOS FROM EMPL WHERE UPPER(APELLIDO) LIKE 'S%';  
DBMS_OUTPUT.PUT_LINE('HAY ' || CUANTOS || ' EMPLEADOS QUE EMPIEZAN POR S');  
  
SELECT EMP_NO, SALARIO+NVL(COMISION,0) INTO NUM, SUELDO  
FROM EMPL WHERE UPPER(APELLIDO) LIKE 'S%';  
DBMS_OUTPUT.PUT_LINE('HAY REGISTROS ' || SQL%ROWCOUNT);  
DBMS_OUTPUT.PUT_LINE('EL NUMERO DE EMPLEADO DEL QUE EMPIEZA POR S ES ' || NUM);  
DBMS_OUTPUT.PUT_LINE('EL SUELDO DEL EMPLEADO ES ' || SUELDO);  
EXCEPTION  
  WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('NO HAY DATOS');  
  WHEN TOO_MANY_ROWS THEN DBMS_OUTPUT.PUT_LINE(' HAY MAS DE UNA FILA !!');  
END;  
/
```

# Actividades I

- Actividad 1. Escribe un bloque PL/SQL que muestre el salario y la comisión del empleado cuyo número es 7902
- Actividad 2. Muestre todos los datos del cliente con número es 7902, con el siguiente formato:

El empleado con número... tiene un salario de ...  
y una comisión de....

# Actividad1 SOL

```
/* . Escribe un bloque PL/SQL que muestre el salario y la comisión del  
    empleado  
    cuyo número es 7902 */  
DECLARE  
VnumEmple emple.emp_no%type;  
Vsalario emple.salario%type;  
Vcomision emple.comision%type;  
BEGIN  
select emp_no,salario,comision INTO VnumEmple,Vsalario,Vcomision from  
    emple WHERE emp_no=7902;  
DBMS_OUTPUT.PUT_LINE('EMPLEADO NUMERO : ' || VnumEmple);  
DBMS_OUTPUT.PUT_LINE('COMISION : ' || Vcomision);  
DBMS_OUTPUT.PUT_LINE('SALARIO : ' || Vsalario);  
END;
```

# Actividad2 SOL

/\*Muestre todos los datos del cliente con número es 7902, con el siguiente formato: El empleado con número... tiene un salario de ... y una comisión de...\*/

DECLARE

VnumEmple emple.emp\_no%type;

Vsalario emple.salario%type;

Vcomision emple.comision%type;

BEGIN

select emp\_no,salario,comision INTO VnumEmple,Vsalario,Vcomision from emple  
WHERE emp\_no=7902;

DBMS\_OUTPUT.PUT\_LINE('EMPLEADO NUMERO : ' || VnumEmple || ' tiene un salario  
de ' || Vsalario || ' y una comision de ' || Vcomision);

END;

```
SQL> DECLARE
  2  numEmple emp.empno%type;
  3  salario emp.sal%type;
  4  comision emp.comm%type;
  5  BEGIN
  6  select empno,sal,comm INTO numEmple,salario,comision from emp WHERE empno=7
902;
  7  DBMS_OUTPUT.PUT_LINE('EMPLEADO NUMERO : ' || numEmple || 'tiene un salario de
' || salario || 'y una comision de ' || comision);
  8  end;
  9  /
EMPLEADO NUMERO : 7902tiene un salario de 3000y una comision de
PL/SQL procedure successfully completed.
```

# Actividad2 SOL

```
set serveroutput on;
DECLARE
Vemple emple%rowtype;
BEGIN
select * INTO Vemple from emple WHERE emp_no=7000;
DBMS_OUTPUT.PUT_LINE('EMPLEADO NUMERO : '||Vemple.emp_no||'
    tiene un salario de '||Vemple.salario||' y una comision de '
    ||nvl(Vemple.comision,0));
EXCEPTION
    WHEN No_data_found THEN
        DBMS_OUTPUT.PUT_LINE('No hay empleados con esas características');
    when others then
        DBMS_OUTPUT.PUT_LINE('Codigo de Error '||sqlcode);
        DBMS_OUTPUT.PUT_LINE('Error '||sqlerrm);
END;
/
```

# Control del flujo del programa

- En PL/SQL es posible ejecutar un bloque de instrucciones si se cumple una expresión lógica:

```
IF expresión_lógica THEN
    instrucciones; -- Si la expresión es cierta
END IF;
```

- Si hay que realizar una acción en caso de que no se cumpla la condición de la expresión, se amplía:

```
IF expresión_lógica THEN
    instrucciones; -- Si la expresión es cierta
ELSE
    instrucciones; -- Si la expresión es falsa
END IF;
```

# Ejemplo if ... else

```
/*Muestre todos los datos del cliente con número es 7902, con el siguiente formato: El empleado con
   Número... tiene un salario de ... y una comisión de.... Si el sueldo es menor a 2000 mostrará el mensaje ..*/
DECLARE
VnumEmple emple.emp_no%type;
Vsalario emple.salario%type;
Vcomision emple.comision%type;
BEGIN
select emp_no,salario,comision INTO VnumEmple,Vsalario,Vcomision from emple WHERE emp_no=7369;
DBMS_OUTPUT.PUT_LINE('EMPLEADO NUMERO : ' || VnumEmple || ' tiene un salario de ' || Vsalario || ' y una
   comisión de ' || Vcomision);
if Vsalario <2000
then
DBMS_OUTPUT.PUT_LINE('El salario es bajo!!!');
Elsif Vsalario <2000 then
DBMS_OUTPUT.PUT_LINE('El salario es MENOR A 2000 !!!');
Else
   DBMS_OUTPUT.PUT_LINE('El salario es igual A 2000 !!!');
end if;

EXCEPTION
   WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('NO HAY DATOS');
END;
/
```

# Control del flujo del programa

- En caso de que se pretenda ejecutar un bloque u otro en función de distintas expresiones, se puede ampliar la estructura de la siguiente forma:

```
IF expr_1 THEN
    bloque;--Si expr_1 es cierta
ELSIF expr_2 THEN
    bloque;--Si expr_1 es falsa y expr_2 cierta
ELSIF expr_3 THEN
    bloque;--Si 1 y 2 son falsas y 3 es cierta
ELSE
    bloque;--Si todas las anteriores son falsas
END IF;
```



# IF Compuesto

```
DECLARE
numEmple emp.empno%type;
salario emp.sal%type;
comision emp.comm%type;
BEGIN
select empno,sal,comm INTO numEmple,salario,comision from emp WHERE empno=7902;
DBMS_OUTPUT.PUT_LINE('EMPLEADO NUMERO : ' || numEmple || ' tiene un salario de ' || salario || ' y
    una comision de ' || comision);
if salario <=1000 then
    DBMS_OUTPUT.PUT_LINE('El salario es muy bajo');
elsif salario>1000 and salario <=2000 then
    DBMS_OUTPUT.PUT_LINE('El salario es bajo');
else
    DBMS_OUTPUT.PUT_LINE('El salario es mayor a 2000');
end if;
END;
```

# Alternativa múltiple (case)

```
CASE [ expresión ]  
    WHEN {condición1|valor1} THEN  
        SENTENCIAS1;  
    WHEN {condición2|valor2} THEN  
        SENTENCIAS2;  
    ...  
    ELSE  
        SENTENCIAS_POR_DEFECTO;  
END CASE;
```

# Alternativa múltiple (case)

```
SET SERVEROUTPUT ON;
DECLARE
NUM NUMBER;
BEGIN
NUM:=&numeroDIA;
CASE num
WHEN 1 THEN
    DBMS_OUTPUT.PUT_LINE('Lunes');
WHEN 2 THEN
    DBMS_OUTPUT.PUT_LINE('martes');
WHEN 3 THEN
    DBMS_OUTPUT.PUT_LINE('miercoleS');
ELSE
    DBMS_OUTPUT.PUT_LINE('No es un dia de la semana! :');
END CASE;
END;
/
```

# Alternativa múltiple (case)

```
SET SERVEROUTPUT ON;
DECLARE
NUM NUMBER;
BEGIN
NUM:=&numeroDIA;
CASE
WHEN num=1 THEN
  DBMS_OUTPUT.PUT_LINE('Lunes');
WHEN num=2 THEN
  DBMS_OUTPUT.PUT_LINE('martes');
WHEN num=3 THEN
  DBMS_OUTPUT.PUT_LINE('miercoleS');
ELSE
  DBMS_OUTPUT.PUT_LINE('No es un dia de la semana! :');
END CASE;
END;
/
```

# Actividades II

## Ej1 :

Pide un valor de radio con decimales y obtén la superficie del círculo. Declara el valor de pi como una constante.

## Ej 2:

Pide un nombre y un apellido y muestra el resultado de su concatenación en mayúsculas , así como la longitud del resultado concatenado

## Ej3 :

Pide al usuario un número y si es par, muestre un mensaje que lo indique

## Ej4:

Repite el ejercicio anterior, indicando si es par o impar

## Ej5:

Ampliamos el anterior e indicamos si es par, si es impar, si es múltiplo de 3 o no es múltiplo de 3.

## Ej6:

Lee una nota numérica e indica si es un Suspenso, Suficiente, etc.

# Ejercicio1

```
/*Pide un valor de radio con decimales y obtén la superficie  
  del círculo. Declara el valor de pi como una constante.*/  
declare  
  pi constant number(6,5):= 3.14159;  
  rad number (2) := &radio;  
  area number(8,2) ;  
begin  
  area:=pi*rad**2;  
  dbms_output.put_line('El área es '||area);  
end;
```

# Ejercicio 2

```
/*Pide un nombre y un apellido y muestra el resultado de su
concatenación en mayúsculas , así como la longitud del resultado
concatenado*/
declare
nombre varchar2(10):='&NOMBRE';
apellido varchar2(10):='&APELLIDO';
concatena varchar2(30);
longitud number(3);
begin
concatena:=upper(nombre || ' ' || apellido);
longitud:=length(concatena);
dbms_output.put_line(concatena || ' ' || longitud);
end;
```

# Ejercicio 3

```
/*Pide al usuario un número y si es par, muestre un  
  mensaje que lo indique*/  
declare  
  valor number(3):=&n;  
begin  
  if mod(valor,2) =0 then  
    dbms_output.put_line('El número ' || valor || ' es par');  
  end if;  
end;
```



# Ejercicio 4

```
/*Completamos el ejercicio anterior indicando si es par o
   impar*/
declare
  valor number(3):=&n;
begin
  if mod(valor,2) =0 then
    dbms_output.put_line('El número ' || valor || ' es par');
  else
    dbms_output.put_line('El número ' || valor || ' es impar');
  end if;
end;
```

# Ejercicio 5

```
/*Ampliamos el anterior e indicamos si es par, si es impar, si es múltiplo de 3 o no es múltiplo de 3.*/  
declare  
    numero number(3);  
begin  
    numero:=&valor;  
    if mod(numero,2)=0 then  
        if mod(numero,3)=0 then  
            dbms_output.put_line('El numero es par y múltiplo de 3');  
        else  
            dbms_output.put_line('El numero es par y no múltiplo de 3');  
        end if;  
    else  
        if mod(numero,3)=0 then  
            dbms_output.put_line('El numero es impar y múltiplo de 3');  
        else  
            dbms_output.put_line('El numero es impar y no múltiplo de 3');  
        end if;  
    end if;  
end;  
end;
```

# Ejercicio 5B

```
declare
  numero number(3);
begin
  numero:=&valor;
  if mod(numero,2)=0 and mod(numero,3)=0 then
    dbms_output.put_line('El numero es par y múltiplo de 3');
  else
    if mod(numero,2)=0 and mod(numero,3)!=0 then
      dbms_output.put_line('El numero es par y no múltiplo de 3');
    else
      if mod(numero,2)!=0 and mod(numero,3)=0 then
        dbms_output.put_line('El numero es impar y múltiplo de 3');
      else
        dbms_output.put_line('El numero es impar y no múltiplo de 3');
      end if;
    end if;
  end if;
end;
```

# Ejercicio 6

```
set serveroutput on;
declare
nota number not null default 0;
calificacion varchar2(10);
begin
nota:=&Dame_Nota;
case
when nota>=0 and nota <=4 then
    dbms_output.put_line('INSUFICIENTE');
when nota>=5 and nota <=6 then
    dbms_output.put_line('SUFICIENTE');
when nota>=7 and nota <9 then
    dbms_output.put_line('NOTABLE');
when nota>=9 and nota <=10 then
    dbms_output.put_line('SOBRESALIENTE');
else
    dbms_output.put_line('NOTA INCORRECTA');
end case;
end;
/
```

# Ejercicio 6B

```
set serveroutput on;
declare
nota number not null default 0;
calificacion varchar2(10);
begin
nota:=&Dame_Nota;
case
when nota BETWEEN 0 and 4 then
    dbms_output.put_line('INSUFICIENTE');
when nota BETWEEN 5 and 6 then
    dbms_output.put_line('SUFICIENTE');
when nota BETWEEN 7 and 8 then
    dbms_output.put_line('NOTABLE');
when nota BETWEEN 9 and 10 then
    dbms_output.put_line('SOBRESALIENTE');
else
    dbms_output.put_line('NOTA INCORRECTA');
end case;
end;
/
```

# Ejercicio 6C

```
set serveroutput on;
declare
nota number not null default 0;
calificacion varchar2(10);
begin
nota:=&Dame_Nota;
case
when nota IN (0,1,2,3,4) then
    dbms_output.put_line('INSUFICIENTE');
when nota IN (5,6) then
    dbms_output.put_line('SUFICIENTE');
when nota IN ( 7, 8) then
    dbms_output.put_line('NOTABLE');
when nota IN ( 9, 10 ) then
    dbms_output.put_line('SOBRESALIENTE');
else
    dbms_output.put_line('NOTA INCORRECTA');
end case;
end;
/
```

# Actividades III

EJ 7: Dados 2 números, calcula el resultado de dividir el mayor de ambos entre el menor. Si el menor es 0, emite un mensaje que indique que no se puede hacer la división.

EJ 8: Dadas 3 letras, muestra un mensaje que las contenga ordenadas alfabéticamente.

EJ9: Averigua si un año dado es bisiesto. Todos los años múltiplos de 4 son bisiestos, salvo los que son múltiplos de 100 y no de 400.

EJ10: Leer el valor de dos variables, que es el resultado de un partido de fútbol, e.d., los goles del equipo que juega en casa y los del equipo que juega fuera, tras la lectura tiene que mostrar por pantalla el resultado y el signo de la quiniela.

# Ejercicio 7

/\*     Dados 2 números, calcula el resultado de dividir el mayor de ambos entre el menor.  
Si el menor es 0, emite un mensaje que indique que no se puede hacer la división.\*/

```
declare
  valor1 number(3);
  valor2 number(3);
  resultado number(3);
begin
  valor1:=&valor;
  valor2:=&valor;
  if valor1>valor2 then
    if valor2!=0 then
      resultado:=valor1/valor2;
      dbms_output.put_line ('El resultado es:' || resultado);
    else
      dbms_output.put_line ('No se puede dividir entre 0');
    end if;
  else
    if valor1!=0 then
      resultado:=valor2/valor1;
      dbms_output.put_line ('El resultado es:' || resultado);
    else
      dbms_output.put_line ('No se puede dividir entre 0');
    end if;
  end if;
end;
```



# Ejercicio 8

/\*- Dadas 3 letras, muestra un mensaje que las contenga ordenadas alfabéticamente.\*/

```
SET SERVEROUTPUT ON;
DECLARE
L1 VARCHAR2(1);
L2 VARCHAR2(1);
L3 VARCHAR2(1);
BEGIN
L1:='&Letra1';
L2:='&Letra2';
L3:='&Letra3';

if L1<L2 and L2<L3 or L1 = L2 AND L1<L3 then
  dbms_output.put_line ('Las letras ordenadas son '||L1||', '||L2||', '||L3);
elsif L1<L3 and L3<L2 OR L1 = L3 AND L1<L2 then
  dbms_output.put_line ('Las letras ordenadas son '||L1||', '||L3||', '||L2);
elsif L2<L1 and L1<L3 OR L1 = L2 AND L1<L3 then
  dbms_output.put_line ('Las letras ordenadas son '||L2||', '||L1||', '||L3);
elsif L2<L3 and L3<L1 OR L2 = L3 AND L2<L1 then
  dbms_output.put_line ('Las letras ordenadas son '||L2||', '||L3||', '||L1);
elsif L3<L1 and L1<L2 OR L1 = L3 AND L1<L2 then
  dbms_output.put_line ('Las letras ordenadas son '||L3||', '||L1||', '||L2);
elsif L3<L2 and L2<L1 OR L3 = L2 AND L2<L1 then
  dbms_output.put_line ('Las letras ordenadas son '||L3||', '||L2||', '||L1);
else
  dbms_output.put_line ('Las letras son iguales');
end if;
end;
/
```

# Ejercicio 8B

/\*- Dadas 3 letras, muestra un mensaje que las contenga ordenadas alfabéticamente.\*/

```
SET SERVEROUTPUT ON;
DECLARE
L1 VARCHAR2(1);
L2 VARCHAR2(1);
L3 VARCHAR2(1);
BEGIN
L1:='&Letra1';
L2:='&Letra2';
L3:='&Letra3';

if L1<L2 and L2<L3 or L1 = L2 AND L1<L3 then
  dbms_output.put_line ('Las letras ordenadas son '||L1||', '||L2||', '||L3);
else
if L1<L3 and L3<L2 OR L1 = L3 AND L1<L2 then
  dbms_output.put_line ('Las letras ordenadas son '||L1||', '||L3||', '||L2);
else
if L2<L1 and L1<L3 OR L1 = L2 AND L1<L3 then
  dbms_output.put_line ('Las letras ordenadas son '||L2||', '||L1||', '||L3);
else
if L2<L3 and L3<L1 OR L2 = L3 AND L2<L1 then
  dbms_output.put_line ('Las letras ordenadas son '||L2||', '||L3||', '||L1);
else
if L3<L1 and L1<L2 OR L1 = L3 AND L1<L2 then
  dbms_output.put_line ('Las letras ordenadas son '||L3||', '||L1||', '||L2);
else
if L3<L2 and L2<L1 OR L3 = L2 AND L2<L1 then
  dbms_output.put_line ('Las letras ordenadas son '||L3||', '||L2||', '||L1);
else
  dbms_output.put_line ('Las letras son iguales');
end if;
end if;
end if;
end if;
end if;
end;
/
```

# Ejercicio 9

/\*Averigua si un año dado es bisiesto. Todos los años múltiplos de 4 son bisiestos, salvo los que son múltiplos de 100 y no de 400.\*/

DECLARE

    NUMERO1 NUMBER(10);

BEGIN

    NUMERO1:=&AÑO;

    IF mod(NUMERO1, 4)=0 THEN

        IF MOD(NUMERO1, 100)=0 THEN

            IF NOT MOD(NUMERO1, 400)=0 THEN

                dbms\_output.put\_line('NO ES BISIESTO');

            ELSE

                dbms\_output.put\_line('ES BISIESTO');

            END IF;

        ELSE

            dbms\_output.put\_line('ES BISIESTO');

        END IF;

    ELSE

        dbms\_output.put\_line('NO ES BISIESTO');

    END IF;

END;

# Ejercicio 10

```
set serveroutput on;
declare
eqlocal number not null default 0;
eqinvitado number;
resultado char;
begin
eqlocal:=&Dame_Goles_locales;
eqinvitado:=&Dame_Goles_invitado;
if eqlocal<eqinvitado then
    resultado:='2';
elsif eqlocal>eqinvitado then
    resultado:='1';
else
    resultado:='x';
end if;
dbms_output.put_line('El resultado es: ' || resultado || ' Goles EQ. LOCAL ' || EQLOCAL || ' Goles EQ.
INVITADO ' || EQINVITADO);
end;
/
```

# BUCLES

- Los bucles repiten una sentencia/instrucción o grupo de sentencias varias veces. Hay varios tipos de bucles, dependiendo del problema a resolver será más idóneo utilizar uno u otro. Si se conoce de antemano el número de iteraciones, se utilizará un FOR. Si se quiere salir del bucle cuando se cumpla una condición, se utilizará LOOP. Se utilizará WHILE cuando no se conozca el número de iteraciones y la condición vaya al principio.

# Bucles

- **FOR**

Repite las sentencias un número fijo de veces.

No es necesario declarar la variable que hace de contador del bucle

Si se añade REVERSE el bucle irá hacia atrás

```
FOR ÍNDICE IN [REVERSE] VALOR_INICIAL..VALOR_FIN AL  
LOOP  
    Instrucciones;  
END LOOP;
```

# BUCLE FOR

```
SET SERVEROUTPUT ON;  
DECLARE  
NUM NUMBER;  
BEGIN  
FOR num in 1..10  
LOOP  
    DBMS_OUTPUT.PUT_LINE(NUM);  
END LOOP;  
END;  
/
```

# BUCLE WHILE

- El bucle se ejecuta mientras la condición sea verdadera, no se conoce el número de iteraciones.

```
WHILE condición LOOP  
    Instrucción/es;  
END LOOP;
```



# BUCLE WHILE

```
SET SERVEROUTPUT ON;  
DECLARE  
NUM NUMBER:=0;  
BEGIN  
WHILE NUM <10  
LOOP  
DBMS_OUTPUT.PUT_LINE(NUM);  
NUM:=NUM+1;  
END LOOP;  
END;  
/
```

# BUCLE WHILE

```
SET SERVEROUTPUT ON;  
DECLARE  
NUM NUMBER:=&numero;  
BEGIN  
WHILE NUM >0  
LOOP  
DBMS_OUTPUT.PUT_LINE(NUM);  
NUM:=NUM-1;  
END LOOP;  
END;  
/
```

# BUCLE ITERAR

- Iterar... fin iterar salir si ...

```
LOOP  
/* Sentencias; */  
EXIT WHEN CONDICIÓN; /* Condición de salida */  
END LOOP;
```

EXIT puede ser utilizado para interrumpir cualquier bucle, de dos formas:

```
EXIT WHEN CONDICIÓN;
```

```
IF CONDICIÓN THEN  
    EXIT  
END IF
```

# BUCLE ITERAR

```
SET SERVEROUTPUT ON;  
DECLARE  
NUM NUMBER:=&numero;  
BEGIN  
LOOP  
DBMS_OUTPUT.PUT_LINE(NUM);  
NUM:=NUM-1;  
EXIT WHEN NUM=0;  
END LOOP;  
END;  
/
```

# BUCLE ITERAR

```
SET SERVEROUTPUT ON;  
DECLARE  
NUM NUMBER:=&numero;  
BEGIN  
LOOP  
DBMS_OUTPUT.PUT_LINE(NUM);  
NUM:=NUM-1;  
IF NUM=0 THEN  
    EXIT;  
END IF;  
END LOOP;  
END;  
/
```

# ACTIVIDADES IV

- EJ1: Mostrar por pantalla los números pares menores de 100 y su suma. Utiliza los tres tipos de bucles.
- EJ2: Lee un número y visualiza su tabla de multiplicar.
- EJ3: Mostrar por pantalla las tablas de multiplicar del 1 al 10.

# ACTIVIDADES IV

**EJ4:** ¿Cuántas veces se ejecutará la instrucción de visualización?

```
SET SERVEROUTPUT ON;
DECLARE
N CONSTANT NUMBER:=10;
BEGIN
FOR I IN 1..N
LOOP
FOR J IN 1..N
LOOP
    DBMS_OUTPUT.PUT('*');
END LOOP;
    DBMS_OUTPUT.PUT_line("");
END LOOP;
END;
/
```

# ACTIVIDADES IV

**EJ5:** ¿Cuántas veces se ejecutará la instrucción de visualización?

```
SET SERVEROUTPUT ON;
DECLARE
N CONSTANT NUMBER:=10;
BEGIN
FOR I IN 1..N
LOOP
    FOR J IN 1..N-I
    LOOP
        DBMS_OUTPUT.PUT('*');
    END LOOP;
    DBMS_OUTPUT.PUT_line("");
END LOOP;
END;
/
```



# EXCEPCIONES

- PL/SQL puede tratar los errores producidos durante la ejecución de un programa que impiden que el programa llegue a su fin utilizando EXCEPCIONES.
- Existen excepciones estándar de PL/SQL que podemos utilizar siempre que lo necesitemos, aunque siempre es recomendable intentar eliminar el error antes de que se lance una excepción, ya que está termina el bloque. También se pueden crear excepciones y lanzarlas con el comando RAISE.
- Las excepciones se colocan en un bloque propio justo antes del END de fin del módulo:

```
BEGIN
/*Sentencias ;*/
EXCEPTION
/*Excepciones ;*/
WHEN nombre_excepcion THEN Sentencias a ejecutar;
END;
```

# EXCEPCIONES

Las excepciones más comunes son:

- **NO\_DATA\_FOUND**: Un select no devuelve ningún resultado en la búsqueda.
- **TOO\_MANY\_ROWS** : Un select devuelve más de una fila
- **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\_CURSOR**: Error de operación sobre un cursor.
- **INVALID\_NUMBER**: La conversión de una cadena a un número no puede realizarse, porque la cadena no representa un valor numérico válido.
- **DUP\_VAL\_ON\_INDEX**: En un insert se intenta insertar un campo clave o único ya existente

# EXCEPCIONES

- **ZERO\_DIVIDE:** Se intenta dividir entre 0.
- **VALUE\_ERROR:** Se ha introducido un valor inválido en un campo, por ejemplo, un valor de más tamaño que lo que admite la variable.
- **LOGIN\_DENIED:** Un programa está intentando acceder a la BD con un usuario o password incorrectos.
- **NOT\_LOGGED\_ON:** Un programa intenta ejecutarse sin realizar la conexión.
- **TIMEOUT\_ON\_RESOURCE:** Se ha terminado el tiempo que SGBD puede esperar por algún recurso.
- **OTHERS:** Cualquier otro error que pudiera surgir

# EXCEPCIONES

- Cada una de estas excepciones, lanza su propio número de error y tiene un mensaje asociado. **SQLCODE** es siempre es un número negativo y **SQLERRM** Mensaje estándar para ese error
- Cuando ocurre una excepción podemos ejecutar varias sentencias como mostrar mensajes, realizar sentencias DML (insert, update o delete) o simplemente null, si no queremos que haga nada.
- En general, sea una excepción propia, lanzada con RAISE o simplemente porque se produce un error estándar contemplado en el bloque de excepciones, la ejecución saltará a la excepción correspondiente, realizará las sentencias que le hayamos asociado y terminará el programa. Funcionaría como un EXIT general del programa dándonos la oportunidad antes de realizar una serie de operaciones

# Excepciones lanzadas por el Programador

- Con la sentencia RAISE, se puede lanzar una excepción de forma explícita. Se puede utilizar en cualquier lugar de la ejecución, pero antes ha de ser declarada en la sección DECLARE:

```
DECLARE
Nombre_excepcion EXCEPTION;
BEGIN
* /... RAISE nombre_excepcion..*/
EXCEPTION
WHEN nombre_excepcion_ THEN
/* Sentencias asociadas */
END;
```

# Excepciones lanzadas por el Programador

```
/* Creación de una excepción propia para tratar los valores negativos */
set serveroutput on;
DECLARE
    num number:=&DameNumero;
    NEGATIVOS Exception;
BEGIN
    if num <0 then raise NEGATIVOS;
    END IF;
    dbms_output.put_line('numero ' || num);
EXCEPTION
    WHEN NEGATIVOS THEN dbms_output.put_line('Numero NEGATIVO!!!');
    WHEN VALUE_ERROR THEN dbms_output.put_line('error DEMASIADO GRANDE!!!');
    WHEN OTHERS THEN dbms_output.put_line('Otro error ' || SQLERRM);
END;
/
```

# Creación de Funciones y Procedimientos

- Para crear una función o procedimiento se pulsa con botón derecho sobre el icono “Functions” o “Procedures” del navegador de objetos y se elige la opción “New function” o “New Procedure” respectivamente.
- Se introduce el nombre del procedimiento o función, los nombres de los parámetros, sus tipos de datos, el modo del parámetro y los valores por defecto. Para el caso de las funciones también hay que especificar el tipo del resultado de la función (parámetro <Return>):
- El asistente abre una pestaña de edición con el código generado para la función o el procedimiento con la cabecera especificada y el cuerpo vacío.

# Función

- Una función es un código compilado en el servidor, que se ejecutan en local, y que pueden aceptar parámetros de entrada y tiene un solo parámetro de salida. Una función SIEMPRE debe devolver un valor.
- Una función se puede usar en otro código PL/SQL, o en SQL ya sea en un SELECT, una clausula WHERE, CONNECT BY, START WITH, ORDER BY, GROUP BY, como VALUES en un INSERT, o como SET en un UPDATE.
- PL/SQL tiene un gran número de funciones incorporadas y permite definir nuevas funciones al usuario.



# Funciones definidas por el usuario

```
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 excepcion
END [<fn_name>];
```

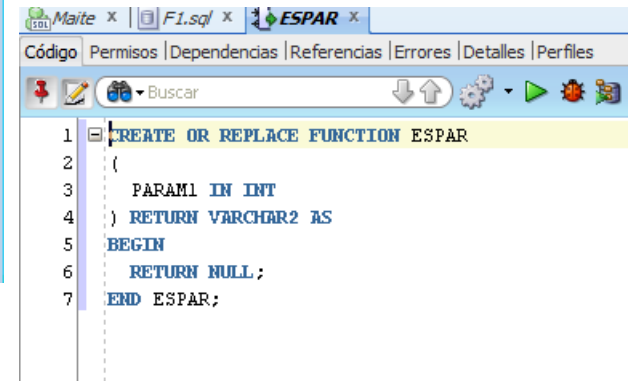
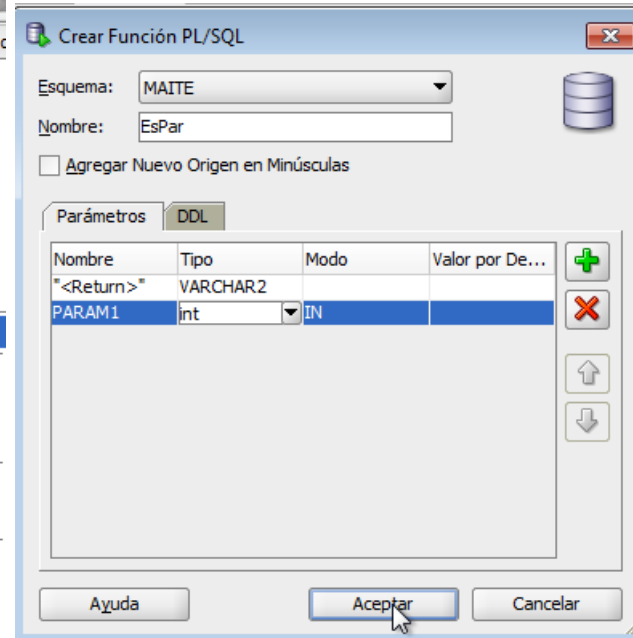
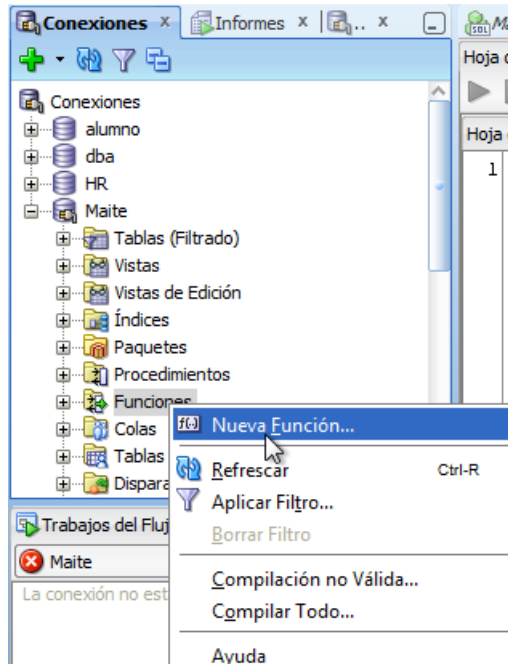
La sintaxis de los parámetros es la misma que en los procedimientos almacenados.

# Ejemplo de función

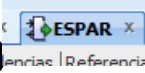

```
create table emp as select * from employee;
CREATE OR REPLACE
FUNCTION FObtenerSalario (p_empno NUMBER)
RETURN NUMBER
IS
result NUMBER;
BEGIN
SELECT salary INTO result FROM employee WHERE emp_no = p_empno;
return(result);
EXCEPTION
WHEN NO_DATA_FOUND THEN
return 0;
END ;
```

Si el sistema nos indica que la función se ha creado con errores de compilación podemos ver estos errores de compilación con la orden SHOW ERRORS en SQL\*Plus.

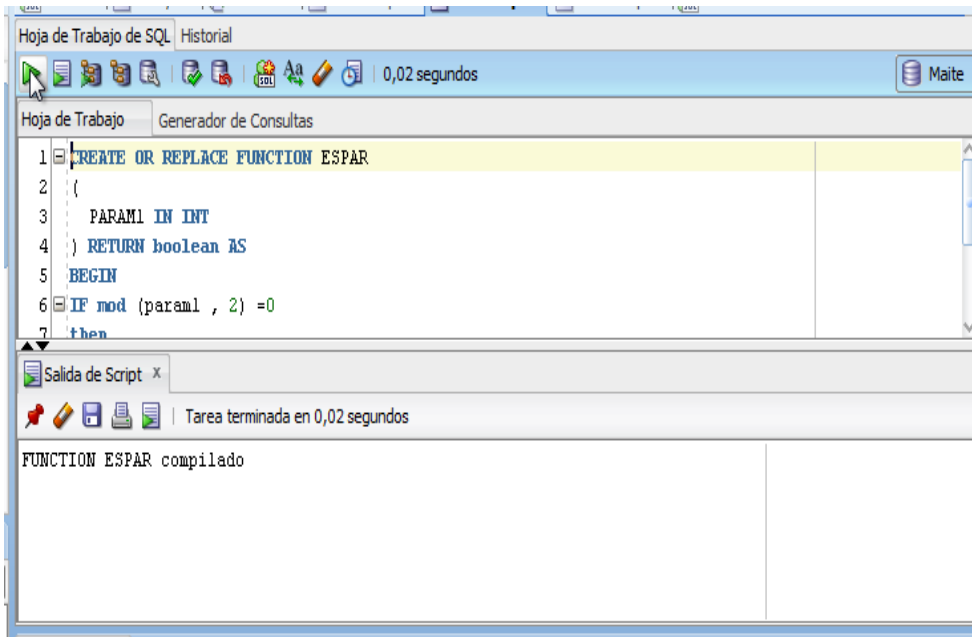
# Creación de Funciones




# Funciones

- Para compilar se pulsa el icono  También se compila automáticamente cuando se almacena el procedimiento o función en la base de datos (icono  ). Los errores y warnings aparecen en el panel “Log” en la pestaña “Compiler”. Junto a la palabra error o warning se indica entre paréntesis la línea y la columna en la que se ha producido el error. Las sentencias erróneas aparecen subrayadas en rojo y los warnings subrayados en amarillo en la ventana de edición

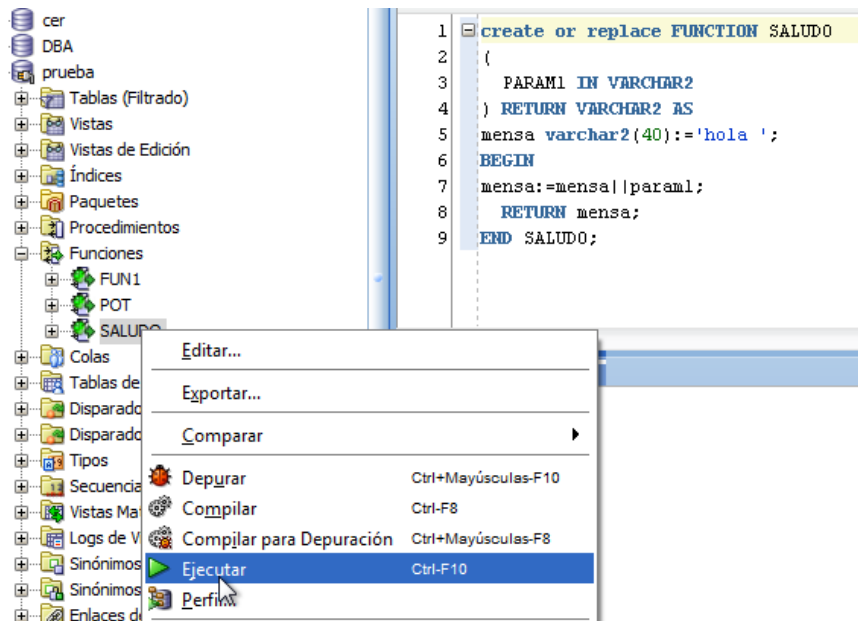
# Compilación desde SQLDeveloper



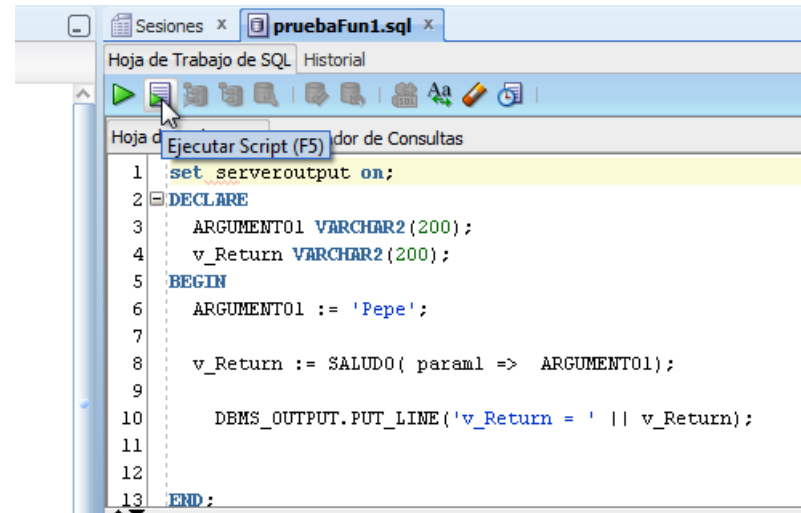
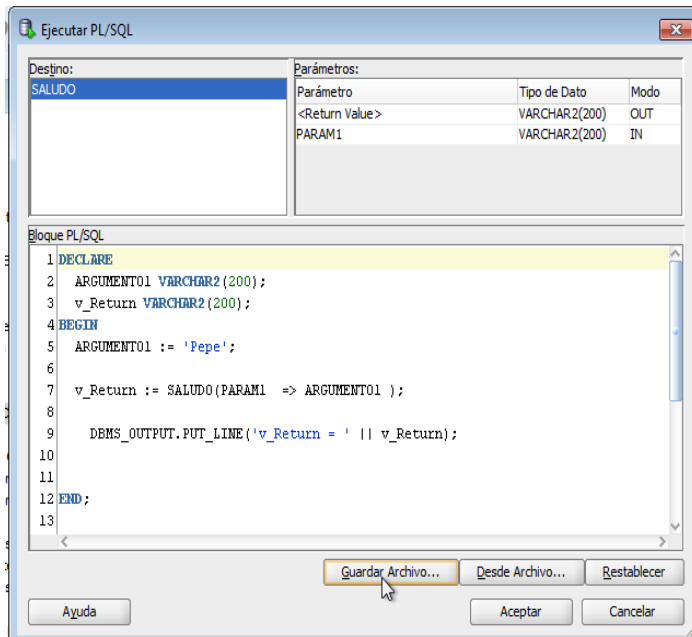
Para ejecutar un procedimiento o función se pulsa el icono  de la ventana de edición o se elige la opción “Run...” que aparece tras pulsar con el botón derecho sobre el icono de la función o procedimiento en el navegador de objetos:

# Ejecución desde SQLDeveloper

- Para poder ejecutar una función o procedimiento, SQL Developer crea un bloque con las variables necesarias para pasar los parámetros en la llamada a la función o procedimiento, debiéndose sustituir los valores por defecto predefinidos, por el valor actual que se le quiere dar al parámetro para la ejecución:



# Ejecución desde SQLDeveloper

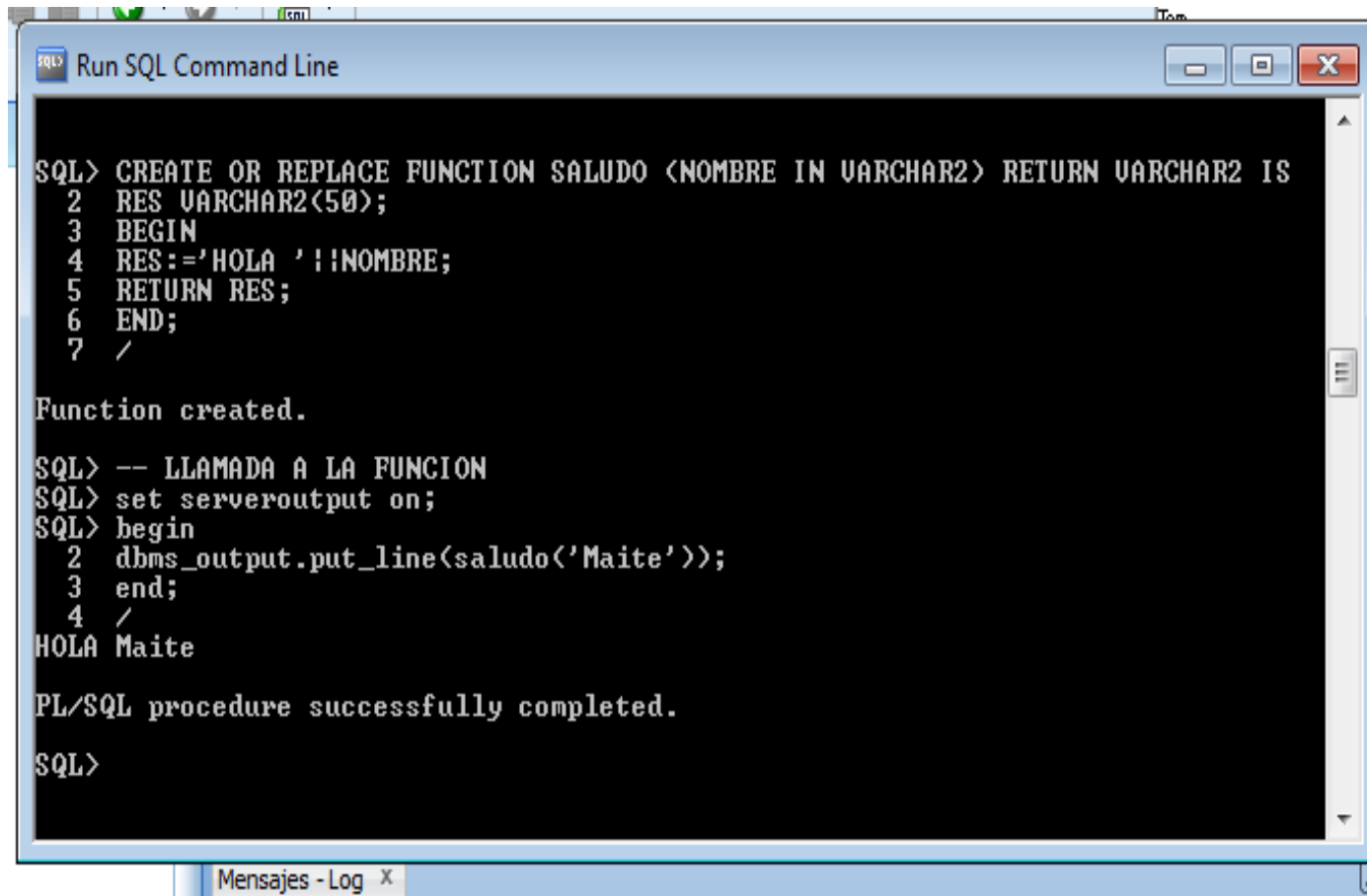


Una vez creado el programa principal y guardado, solo queda ejecutar el script.

---

```
bloque anónimo terminado
v_Return = hola Pepe
```

# Ejecución desde la línea de comandos



```
SQL> CREATE OR REPLACE FUNCTION SALUDO (NOMBRE IN VARCHAR2) RETURN VARCHAR2 IS
  2 RES VARCHAR2(50);
  3 BEGIN
  4 RES:='HOLA '||NOMBRE;
  5 RETURN RES;
  6 END;
  7 /

Function created.

SQL> -- LLAMADA A LA FUNCION
SQL> set serveroutput on;
SQL> begin
  2 dbms_output.put_line(saludo('Maite'));
  3 end;
  4 /
HOLA Maite

PL/SQL procedure successfully completed.

SQL>
```

Mensajes - Log x



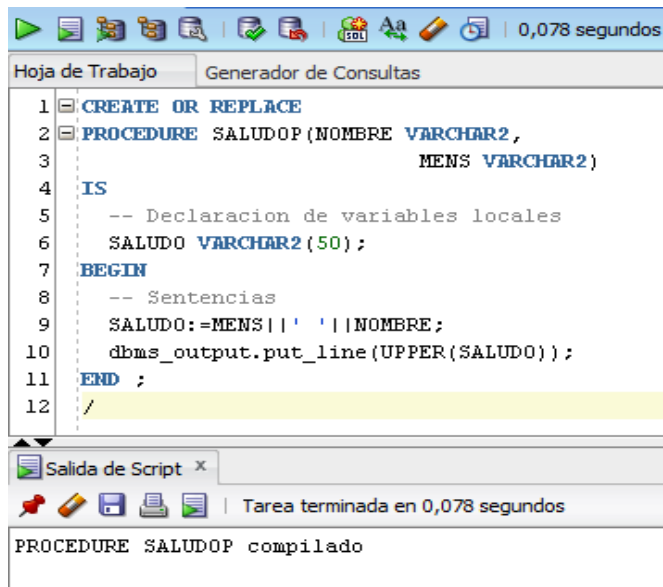
# PROCEDIMIENTOS

- 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.
- La sintaxis de un procedimiento almacenado es la siguiente:

```
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>;
```

# PROCEDIMIENTOS

- Hay especificar el tipo de datos de cada parámetro. Al especificar el tipo de dato del parámetro no debemos especificar la longitud del tipo.
- Los parámetros pueden ser de entrada (**IN**), de salida (**OUT**) o de entrada salida (**IN OUT**). El valor por defecto es **IN**, y se toma ese valor en caso de que no especifiquemos nada.

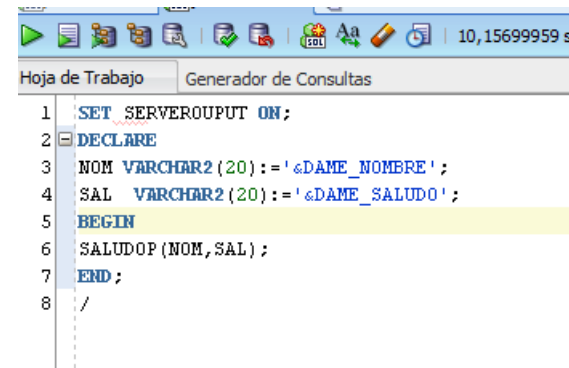


```
1 CREATE OR REPLACE
2 PROCEDURE SALUDOP(NOMBRE VARCHAR2,
3                   MENS VARCHAR2)
4 IS
5     -- Declaracion de variables locales
6     SALUDO VARCHAR2(50);
7 BEGIN
8     -- Sentencias
9     SALUDO:=MENS||' '||NOMBRE;
10    dbms_output.put_line(UPPER(SALUDO));
11 END ;
12 /
```

Salida de Script x

Tarea terminada en 0,078 segundos

PROCEDURE SALUDOP compilado



```
1 SET SERVEROUTPUT ON;
2 DECLARE
3 NOM VARCHAR2(20):='&DAME_NOMBRE';
4 SAL VARCHAR2(20):='&DAME_SALUDO';
5 BEGIN
6 SALUDOP(NOM,SAL);
7 END;
8 /
```

# PROCEDIMIENTOS

- Existen dos formas de pasar argumentos a un procedimiento almacenado a la hora de ejecutarlo. Estas son:
- **Notación posicional:** Se pasan los valores de los parámetros en el mismo orden en que están definidos en procedure.
- **Notación nominal:** Se pasan los valores en cualquier orden nombrando explícitamente el parámetro.

```
SET SERVEROUTPUT ON;  
DECLARE  
  NOM VARCHAR2(20):='&DAME_NOMBRE';  
  SAL VARCHAR2(20):='&DAME_SALUDO';  
BEGIN  
  SALUDOP(NOM, SAL);  
END;  
/
```

```
SET SERVEROUTPUT ON;  
DECLARE  
  NOM VARCHAR2(20):='&DAME_NOMBRE';  
  SAL VARCHAR2(20):='&DAME_SALUDO';  
BEGIN  
  SALUDOP(NOMBRE=>NOM, MENS=>SAL);  
END;  
/
```

# Procedimientos

También podemos asignar un valor por defecto a los parámetros, utilizando la clausula **DEFAULT** o el operador de asignación (:=)

```
create or replace PROCEDURE
  SALUDOP(NOMBRE VARCHAR2,
          MENS VARCHAR2 default 'HOLA')
IS
  -- Declaración de variables locales
  SALUDO VARCHAR2(50);
BEGIN
  -- Sentencias
  SALUDO:=MENS || ' ' || NOMBRE;
  dbms_output.put_line(UPPER(SALUDO));
END;
```

```
SET SERVEROUTPUT ON;
DECLARE
  NOM VARCHAR2(20):='&DAME_NOMBRE';
  -- SAL VARCHAR2(20):='&DAME_SALUDO';
BEGIN
  SALUDOP(NOM);
END;
/
```

# Ejemplo Procedimiento de Intercambio

```
/* PROCEDIMIENTO DE INTERCAMBIO DE DOS VARIABLES
*/
CREATE OR REPLACE PROCEDURE INTERCAMBIO ( VAR1
  IN OUT NUMBER, VAR2 IN OUT NUMBER) IS
AUX NUMBER;
BEGIN
AUX:=VAR1;
VAR1:=VAR2;
VAR2:=AUX;
END;
/
```

# Prueba Procedimiento de Intercambio

```
/* PRUEBA INTERCAMBIO */  
SET SERVEROUTPUT ON;  
DECLARE  
N1 NUMBER:=&DAMENUMERO1;  
N2 NUMBER:=&DAMENUMERO2;  
BEGIN  
DBMS_OUTPUT.PUT_LINE('N1= '||N1||' N2= '||N2);  
INTERCAMBIO(N1,N2);  
DBMS_OUTPUT.PUT_LINE('N1= '||N1||' N2= '||N2);  
END;  
/
```

# Actividades V:

## Ejercicios Funciones y Procedimientos

- EJ1:** Crear un programa anónimo que pida el nombre de un modulo. Si el modulo es FCT tengo que elegir entre APTO y NO APTO y me muestra la elección. Si no es FCT, llama una función ftextoNota a la que le paso una nota y me devuelve si es Supenso, Bien, Notable o Sobresaliente (podéis elegir los rangos de notas). Usad upper() para el nombre del modulo, ya que Oracle es Case Sensitive para las cadenas de texto.
- EJ2:** Crear un procedimiento al que le paso un numero entero y me muestra su tabla de multiplicar (operaciones de 1 al 10 en formato numero x contador=resultado)
- EJ3:** Crear una función fprimo a la que le pase un numero y me diga si es primo o no.

# Actividades V:

## Ejercicios Funciones y Procedimientos

**Ej4:** Crear un Procedimiento **pborraEMPDEPART** que borre los empleados de un departamento pasado por parámetro, visualice el número de filas afectadas y luego visualice los datos del departamento. Desarrollar el programa principal que invoque a este procedimiento.

Finalmente hará un rollback, para que los cambios no se graben.

**Ej5:** Crear un procedimiento pdatosemplenum que pasándole el numero de empleado me muestre nombre, apellidos, oficio, sueldo, numero y nombre de su departamento. Crear un registro especial para recoger los datos con select into.



# Actividades V:

## Ejercicios Funciones y Procedimientos

**EJ6:** Crear un procedimiento pdatosprov al que le paso un código de provincia y me muestra sus valores. Crear un rowtype para recoger los datos. Probar con provincias que no existan y que existen.

**EJ7:** Crear una función fexistepro prov al que le paso un código de provincia y me devuelve 0 si no existe / 1 si existe. Probar con Dual.

**EJ8:** Crear una función fsueldo al que le paso un numero de empleado y me devuelve su salario mas la comisión si esta no es nula (función NVL). Probar a ejecutar la funcion fsueldo en un select para todos los empleados.

# CURSORES

- **Definición Cursor:** Objeto que hace referencia a un conjunto de datos obtenidos de una consulta.
- **Sintaxis de cursores:**

```
-- DECLARAR  un nuevo cursor en la sección DECLARE  
CURSOR cursor_name IS SELECT_statement;  
-- OPEN: Abre el cursor para acceder a sus filas asociadas  
OPEN cursor_name;  
-- FETCH: Extrae la siguiente fila de un cursor  
FETCH cursor_name INTO variable list;  
-- CLOSE: Cierra el cursor  
CLOSE cursor_name ;
```

# CURSORES

- PL/SQL utiliza cursores para gestionar las instrucciones SELECT. Un cursor es un conjunto de registros devuelto por una instrucción SQL. Técnicamente los cursores son fragmentos de memoria que son reservados para procesar los resultados de una consulta SELECT.

Podemos distinguir dos tipos de cursores:

- **Cursores implícitos.** Este tipo de cursores se utiliza para operaciones SELECT INTO, se usa cuando la consulta devuelve un único registro.
- **Cursores explícitos.** Son los cursores que son declarados y controlados por el programador. Se utilizan cuando la consulta devuelve un conjunto de registros. Ocasionalmente también se utilizan en consultas que devuelven un único registro por razones de eficiencia. Son más rápidos.

Un cursor se define como cualquier otra variable de PL/SQL y debe nombrarse de acuerdo a los mismos convenios que cualquier otra variable. Los cursores implícitos no necesitan declaración.

# CURSORES

## Atributos de cursores:

- **SQL%ISOPEN** : True si el cursor esta abierto (entre open y close),
- **SQL%FOUND** :
  - NULL antes de ejecutar.
  - TRUE si uno o mas registros fueron inserted, merged, updated, o deleted o si solo 1 registro fue seleccionado.
  - FALSE si ningún registro fue seleccionado, merged, updated, inserted, o deleted.
- **SQL%NOTFOUND**:
  - NULL antes de ejecutar
  - TRUE si ningún registro fue seleccionado, merged, updated, inserted, o deleted.
  - FALSE si uno o mas registros fueron inserted, merged, updated, deleted o seleccionado.
- **SQL%ROWCOUNT** Cantidad de registros afectados por el cursor.

# EJEMPLO CURSOR EXPLÍCITO1

/\* VISUALIZAR Nombre, Oficio, Salario y Comisión de los empleados del departamento 10 \*/

**DECLARE**

**CURSOR** cursorEmple **IS** SELECT NOMBRE, OFICIO, SALARIO, COMISION FROM EMPLE WHERE  
DEPT\_NO=10 ORDER BY 1;

VNOMBRE EMPLE.NOMBRE%TYPE;

VOFICIO EMPLE.OFICIO%TYPE;

VSALARIO EMPLE.SALARIO%TYPE;

VCOMISION EMPLE.COMISION%TYPE;

**BEGIN**

**OPEN** cursorEmple;

**FETCH** cursorEmple **INTO** VNOMBRE,VOFICIO,VSALARIO,VCOMISION;

**WHILE** cursorEmple%FOUND

**LOOP**

DBMS\_OUTPUT.PUT\_LINE(VNOMBRE || ' ' || VOFICIO || ' ' || VSALARIO || ' ' || NVL(VCOMISION,0));

-- Para evitar errores de formato se convierte el valor numérico a carácter

DBMS\_OUTPUT.PUT\_LINE('EL SUELDO DEL EMPLEADO ES ' || **TO\_CHAR**(VSALARIO+NVL(VCOMISION,0)));

**FETCH** cursorEmple **INTO** VNOMBRE,VOFICIO,VSALARIO,VCOMISION;

**END LOOP;**

**CLOSE** cursorEmple;

**END;**

/

# EJEMPLO CURSOR EXPLÍCITO2

```
/* visualizar TODOS los datos de los empleados del departamento 10 */
```

```
set serveroutput on;
```

```
DECLARE
```

```
CURSOR cursorEmple IS SELECT * FROM EMPLE WHERE DEPT_NO=10 ORDER BY 1;
```

```
VREG emple%rowtype;
```

```
BEGIN
```

```
OPEN cursorEmple;
```

```
FETCH cursorEmple INTO VREG;
```

```
WHILE cursorEmple%FOUND LOOP
```

```
DBMS_OUTPUT.PUT_LINE(VREG.NOMBRE || ' ' || VREG.OFICIO || ' ' || VREG.SALARIO || '  
    ' || NVL(VREG.COMISION,0));
```

```
FETCH cursorEmple INTO VREG;
```

```
END LOOP;
```

```
DBMS_OUTPUT.PUT_LINE('SE HAN PROCESADO ' || cursorEmple%ROWCOUNT || '  
    Registros');
```

```
CLOSE cursorEmple;
```

```
END;
```

```
/
```

# EJEMPLO CURSOR EXPLÍCITO3

/\* visualizar los datos (NOMBRE,OFICIO,SALARIO Y COMISIÒN) de los empleados del departamento 10, utilizando un registro \*/

**DECLARE**

CURSOR cursorEmple IS SELECT NOMBRE,OFICIO,SALARIO,COMISION FROM EMPLE WHERE  
DEPT\_NO=10 ORDER BY 1;

**type REmple IS RECORD** ( VNOMBRE EMPLE.NOMBRE%TYPE, VOFICIO EMPLE.OFICIO%TYPE,  
VSALARIO EMPLE.SALARIO%TYPE, VCOMISION EMPLE.COMISION%TYPE );

VREG REmple;

**BEGIN**

OPEN cursorEmple;

FETCH cursorEmple INTO VREG;

WHILE cursorEmple%FOUND LOOP

DBMS\_OUTPUT.PUT\_LINE(VREG.VNOMBRE||' '||VREG.VOFICIO||' '||VREG.VSALARIO||'  
'|NVL(VREG.VCOMISION,0));

FETCH cursorEmple INTO VREG;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('SE HAN PROCESADO '||cursorEmple%ROWCOUNT||' Registros');

CLOSE cursorEmple;

END;

/

# CURSORES CON ESTRUCTURA FOR

- La utilización de cursores implica:
  - Declarar el cursor.
  - Declarar una variable que recogerá los datos del cursor.
  - Abrir el Cursor.
  - Recuperar con FETCH una a una las filas extraídas, introduciendo los datos en la variable, procesándolos y comprobando también si se han recuperado todos los datos o no.
  - Cerrar el Cursor.
- La estructura de cursores FOR ... LOOP simplifica estas tareas realizándolas todas, excepto la declaración del cursor, de forma implícita.



# CURSORES CON ESTRUCTURA FOR

- 1.- Se declara el cursor en la sección declarativa:

`CURSOR nombreCursor IS sentenciaSelect;`

- 2.- Se procesa el cursor utilizando el siguiente formato:

`FOR nombreVarReg IN nombreCursor LOOP`

`...`

`END LOOP;`

Donde `nombreVarReg` será creada automáticamente por el bucle para recoger los datos del cursor.

Al entrar al bucle se abre el cursor de manera automática.

Se declara implícitamente la variable `nombreVarReg` de tipo `nombreVarReg%rowtype` y se ejecuta un `FETCH` implícito, cuyo resultado quedará en `nombreVarReg`.

A continuación, se realizarán las sentencias hasta llegar al `END LOOP`; que hará volver al `FOR ... LOOP` ejecutando el siguiente `FETCH` implícito...

# Ejemplo de cursor explícito con FOR

```
/* Visualizar el apellido, oficio y comisión de los empleados  
   cuya comisión supera 500 euros*/
```

```
set serveroutput on;
```

```
DECLARE
```

```
CURSOR cursorEmple IS SELECT apellido, oficio, comision  
   FROM emple WHERE comision >500;
```

```
BEGIN
```

```
FOR VREG IN cursorEmple LOOP
```

```
DBMS_OUTPUT.PUT_LINE(VREG.APELLIDO || '  
   ' || VREG.OFICIO || ' ' || NVL(VREG.COMISION,0));
```

```
END LOOP;
```

```
END;
```

# Ejemplo de cursor implícito con FOR

```
DECLARE
remple emple%ROWTYPE;
begin
for var in (select * from emple) loop
dbms_output.put_line(var.nombre);
end loop;
end;
/
```

# Ejemplo de cursor explícito con FOR

```
DECLARE
CURSOR c_emp IS /*CURSOR*/
select nombre, salario from emple;
BEGIN
FOR fila IN c_emp LOOP /*no es necesario definir la
    variable fila, será de tipo %ROW */
dbms_output.put_line(fila.nombre || ' tiene un salario de
    ' || fila.salario);
END LOOP;
END;
/
```

# Actividades VI: Cursores

- **EJ1:** Visualizar el apellido y la fecha de alta de todos los empleados ordenados por fecha de alta. Utilizando un bucle FOR.
- **EJ2:** Utilizando un bucle WHILE.
- **EJ3:** Mostrar el nombre y localización de todos los centros en orden.

# Actividades VI: Cursores

```
CREATE OR REPLACE PROCEDURE P_VERCENTROS IS
--DEFINICIÓN DE VARIABLES TIPO REGISTRO
TYPE LOCALIZA IS RECORD (
NOMBRE CENTROS.NOMCE%TYPE,
CALLE CENTROS.CALLE%TYPE,
CIUDAD CENTROS.CIUDAD%TYPE
);
VREG LOCALIZA;
-- DEFINICIÓN DEL CURSOR
CURSOR CURSOR1 IS
    SELECT NOMCE, CALLE, CIUDAD FROM CENTROS ORDER BY NUMCE;
BEGIN
OPEN CURSOR1;
FETCH CURSOR1 INTO VREG;
WHILE CURSOR1%FOUND LOOP
DBMS_OUTPUT.PUT_LINE(VREG.NOMBRE || ' ESTA EN ' || VREG.CALLE || ' DE ' || VREG.CIUDAD);
DBMS_OUTPUT.PUT_LINE('LLEVO LEIDAS:' || CURSOR1%ROWCOUNT || CHR(13) || CHR(10));
FETCH CURSOR1 INTO VREG;
IF CURSOR1%FOUND THEN DBMS_OUTPUT.PUT_LINE('QUEDAN MAS, PUEDO SEGUIR LEYENDO'); END IF;
IF CURSOR1%NOTFOUND THEN DBMS_OUTPUT.PUT_LINE('NO QUEDAN, NO PUEDO LEER MAS'); END IF;
END LOOP;
CLOSE CURSOR1;
EXCEPTION
WHEN TOO_MANY_ROWS THEN DBMS_OUTPUT.PUT_LINE('DEMASIADAS FILAS');
WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('ERROR INESPERADO');
END;
```

# Actividades VI: Cursores

- EJECUCIÓN DEL PROCEDIMIENTO, DESDE DEVELOPER:

begin

P\_vercentros;

end;

/

EJECUCIÓN DEL PROCEDIMIENTO, DESDE sqlplus:

```
SQL> set serveroutput on;
SQL> exec p_vercentros;
CENTRAL ESTÁ EN CARRETERA DE LA CORUnna KM 25 DE MAJADAHONDA
LLEVO LEIDAS:1
```

```
QUEDAN MAS, PUEDO SEGUIR LEYENDO
SEDE_NORTE ESTÁ EN CALLE DE LA ARMADA 5 DE ZARAGOZA
LLEVO LEIDAS:2
```

```
QUEDAN MAS, PUEDO SEGUIR LEYENDO
SEDE_SUR ESTÁ EN PLAZA COLON S/N DE SEVILLA
LLEVO LEIDAS:3
```

```
QUEDAN MAS, PUEDO SEGUIR LEYENDO
SEDE_CENTRO ESTÁ EN CALLE ALCALA 120 DE MADRID
LLEVO LEIDAS:4
```

```
NO QUEDAN, NO PUEDO LEER MAS
```

```
PL/SQL procedure successfully completed.
```

```
SQL> call p_vercentros();
CENTRAL ESTÁ EN CARRETERA DE LA CORUnna KM 25 DE MAJADAHONDA
LLEVO LEIDAS:1
```

```
QUEDAN MAS, PUEDO SEGUIR LEYENDO
SEDE_NORTE ESTÁ EN CALLE DE LA ARMADA 5 DE ZARAGOZA
LLEVO LEIDAS:2
```

```
QUEDAN MAS, PUEDO SEGUIR LEYENDO
SEDE_SUR ESTÁ EN PLAZA COLON S/N DE SEVILLA
LLEVO LEIDAS:3
```

```
QUEDAN MAS, PUEDO SEGUIR LEYENDO
SEDE_CENTRO ESTÁ EN CALLE ALCALA 120 DE MADRID
LLEVO LEIDAS:4
```

```
NO QUEDAN, NO PUEDO LEER MAS
```

```
Call completed.
```

# ALIAS Y CAMPOS CALCULADOS EN UN CURSOR

- Si en un cursor necesitamos incluir un campo calculado, función de agrupamiento o función almacenada, es necesario darle un alias, con el que se tratará ese valor a partir de que el cursor sea abierto. Los alias dentro de PL/SQL no precisan las comillas dobles del SQL estándar.

```
DECLARE
```

```
CURSOR C1 IS SELECT COUNT(*)  numemple, DEPT_NO from emple  
group by DEPT_NO;
```

```
BEGIN
```

```
FOR v in C1 LOOP
```

```
    DBMS_OUTPUT.PUT_LINE ('EL DEPARTAMENTO ' || v.DEPT_NO || '  
    TIENE ' || v.numemple || ' EMPLEADOS');
```

```
END LOOP;
```

```
END;
```



# Cursor con dos tablas

- Si un cursor debe seleccionar campos de distintas tablas, que coinciden en el nombre, también será necesario utilizar un alias, por ejemplo, en la tabla EMPLE y en la tabla CLIENTES existe un campo nombre, si quisiéramos obtener los dos con un mismo cursor tendríamos que poner alias a los campos :

```
DECLARE
```

```
CURSOR C1 IS SELECT depart.dnombre nombreD, emple.nombre emplenom  
from depart, emple where emple.dept_no=depart.dept_no;
```

```
BEGIN
```

```
FOR v in C1 LOOP
```

```
    DBMS_OUTPUT.PUT_LINE ('EL empleado ' || v.emplenom || ' trabaja en ' ||  
        v.nombreD );
```

```
END LOOP;
```

```
END;
```

```
/
```

# Triggers o disparadores de BD

- Son bloques PL/SQL almacenados que se ejecutan o disparan automáticamente cuando se producen ciertos eventos.
- Hay tres tipos:
  - **Disparadores de tablas.** Se disparan cuando se produce un determinado suceso o evento de manipulación que afecta a la tabla (insertado, borrado o actualización de filas).
  - **Disparadores de sustitución.** Asociados a vistas. Se disparan cuando se intenta ejecutar un comando de manipulación que afecta a la vista.
  - **Disparadores del sistema.** Se disparan cuando se ocurre un evento del sistema (arranque o parada de la BD, entrada o salida de un usuario, etc.) o una instrucción de definición de datos (creación, modificación o eliminación de una tabla u objeto).

# Disparadores de Tablas

- Un trigger es un bloque PL/SQL asociado a una tabla, que se ejecuta como consecuencia de una determinada instrucción SQL (una operación DML: INSERT, UPDATE o DELETE) sobre dicha tabla.
- Un disparador es un código que se lanza cada vez que se ha modificado o se va a modificar el dato de una tabla. Puede ejecutarse a nivel de la consulta, o a nivel de cada línea afectada por la consulta. Puede hacerlo antes o después de la consulta, y solo por ciertos tipos de consulta (insert/update/delete), y eventualmente solo cuando cierto/s campo/s están afectado/s.

# Disparadores o Triggers

La sintaxis para crear un trigger es la siguiente:

```
CREATE [OR REPLACE] TRIGGER <nombre_trigger>
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE [OF col1, col2, ..., colN]
[OR {DELETE | INSERT | UPDATE [OF col1, col2, ..., colN]...}]
ON <nombre_tabla>
[FOR EACH ROW [WHEN (<condicion>)]]
DECLARE
    -- variables locales
BEGIN
    -- Sentencias
[EXCEPTION]
    -- Sentencias control de excepcion
END <nombre_trigger>;
```

# Triggers

- Los triggers pueden definirse para las operaciones INSERT, UPDATE o DELETE, y pueden ejecutarse antes o después de la operación. El modificador BEFORE AFTER indica que el trigger se ejecutará antes o después de ejecutarse la sentencia SQL definida por DELETE INSERT UPDATE. Si incluimos el modificador **OF** el trigger solo se ejecutará cuando la sentencia SQL afecte a los campos incluidos en la lista.
- El alcance de los disparadores puede ser de fila o de orden. El modificador FOR EACH ROW indica que el trigger se disparará cada vez que se realizan operaciones sobre una fila de la tabla. Si se acompaña del modificador WHEN, se establece una restricción; el trigger solo actuará, sobre las filas que satisfagan la restricción.

# Triggers

- El cuerpo de un trigger es un bloque PL/SQL. Cualquier orden que sea legal en un bloque PL/SQL, es legal en el cuerpo de un disparador, con las siguientes restricciones:
  - Un disparador no puede emitir ninguna orden de control de transacciones: **COMMIT**, **ROLLBACK** o **SAVEPOINT**. El disparador se activa como parte de la ejecución de la orden que provocó el disparo, y forma parte de la misma transacción que dicha orden. Cuando la orden que provoca el disparo es confirmada o cancelada, se confirma o cancela también el trabajo realizado por el disparador.
  - Por razones idénticas, ningún procedimiento o función llamado por el disparador puede emitir órdenes de control de transacciones.
  - El cuerpo del disparador no puede contener ninguna declaración de variables LONG o LONG RAW

# Triggers

- **INSERT, DELETE, UPDATE:** Define qué tipo de orden DML provoca la activación del disparador.
- **BEFORE , AFTER** Define si el disparador se activa antes o después de que se ejecute la orden.
- **FOR EACH ROW** Los disparadores con nivel de fila se activan una vez por cada fila afectada por la orden que provocó el disparo. Los disparadores con nivel de orden se activan sólo una vez, antes o después de la orden, independientemente del número de filas afectadas por ella. Se puede incluir la cláusula FOR EACH STATEMENT, aunque no es necesario, se asume por omisión. Los disparadores con nivel de fila se identifican por la cláusula FOR EACH ROW en la definición del disparador.
- El orden en que se comprueban los disparadores es primero los que son BEFORE FOR EACH STATEMENT, luego BEFORE FOR EACH ROW, luego AFTER BEFORE FOR EACH ROW y por último AFTER FOR EACH STATEMENT.

# Triggers

- Dentro del ámbito de un trigger disponemos de las variables OLD y NEW. Estas variables se utilizan del mismo modo que cualquier otra variable PL/SQL, con la salvedad de que no es necesario declararlas, son de tipo **%ROWTYPE** y contienen una copia del registro antes (OLD) y después (NEW) de la acción SQL (INSERT, UPDATE, DELETE) que ha ejecutado el trigger. Utilizando esta variable podemos acceder a los datos que se están insertando, actualizando o borrando.
- Los registros OLD y NEW son sólo válidos dentro de los disparadores con nivel de fila.

La siguiente tabla muestra los valores de OLD y NEW.

ACCION SQL	OLD	NEW
INSERT	No definido; todos los campos toman valor NULL.	Valores que serán insertados cuando se complete la orden.
UPDATE	Valores originales de la fila, antes de la actualización.	Nuevos valores que serán escritos cuando se complete la orden.
DELETE	Valores, antes del borrado de la fila.	No definidos; todos los campos toman el valor NULL.



# Ejemplo Trigger subida de salario

1/ Crearemos una tabla TMODEmple (desde la interfaz gráfica o desde la línea de comando) con un campo VARCHAR2(100) para almacenar los mensajes que queremos guardar cuando se produzca la actualización del sueldo de un empleado.

2/ Código del trigger:

```
CREATE OR REPLACE TRIGGER SUBIDASALARIO
AFTER UPDATE OF SALARIO
ON EMPLE
FOR EACH ROW
BEGIN
INSERT INTO TmodEmple VALUES ('SUBIDA SALARIO EMPLEADO
'||:OLD.EMP_NO);
END;
/
```

# Ejemplo Trigger subida de salario

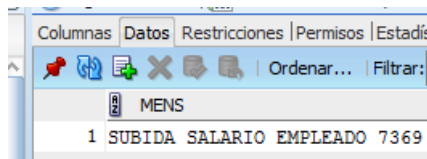
3/ Visualizar el salario del empleado 7369 para luego poder comprobar la actualización.

```
select salario from emple where emp_no=7369;
```

4/ Actualizar el salario de ese empleado en un 10%:

```
update emple set salario=salario+salario*0.10  
where emp_no=7369;
```

5/ Consultar los registros de la tabla TMODEmple y ver que se ha añadido un nuevo mensaje.



# La restricción del trigger

- La cláusula WHEN seguida de una condición restringe la ejecución del trigger al cumplimiento de la condición especificada. Esta condición tiene algunas limitaciones:
- Sólo se puede utilizar con triggers a nivel de fila (FOR EACH ROW)
- Se trata de una condición SQL, no PL/SQL
- No puede incluir una consulta a la misma tabla o a otras tablas o vistas

# Ejemplo Triggers borrado empleado

- Trigger que se disparará cada vez que se elimine un empleado, guardando su numero de empleado, apellido y departamento en la tabla TBorraEmple que se creará previamente con un solo campo VARCHAR2(100).

```
CREATE OR REPLACE TRIGGER BORRAEMPLE
before DELETE
ON EMPLE
FOR EACH ROW
BEGIN
INSERT INTO TbORRAEmple VALUES ('BORRADO EMPLEADO
'||:OLD.EMP_NO||' '||:old.APELLIDO||' DEPARTAMENTO
'||:OLD.DEPT_NO);
END;
/
```

# Ejemplo Triggers borrado empleado

```
SQL>select NOMBRE,APELLIDO from emple  
      where emp_no=7366;
```

```
SQL>DELETE FROM emple where  
      emp_no=7366;
```

-- Comprobaremos que se ha creado un nuevo  
registro en la tabla TBORRAEMPLE.

```
SQL>ROLLBACK;
```

# Triggers

- Cuando se dispara un trigger, éste forma parte de la operación de actualización que lo disparó, de forma que si el trigger falla, Oracle dará por fallida la actualización completa. Aunque el fallo se produzca a nivel de una sola fila, Oracle hará ROLLBACK de toda la actualización.
- Se puede asociar varios triggers a una tabla o utilizar un solo trigger con múltiples eventos de disparo.

# Múltiples eventos de disparo y predicados condicionales

- Un mismo trigger puede ser disparado por distintas operaciones o eventos de disparo. Para indicarlo se utilizará OR.
- Ej. BEFORE DELETE OR UPDATE ON Emple ...
- En estos casos, Oracle permite la utilización de predicados condicionales que devolverán un valor true o false para cada una de las posibles operaciones:
  - **INSERTING** Devuelve TRUE si la orden de disparo es INSERT; FALSE en otro caso.
  - **UPDATING** TRUE si la orden de disparo es UPDATE; FALSE en otro caso.
  - **DELETING** TRUE si la orden de disparo es DELETE; FALSE en otro caso.
  - **UPDATING (nombreColumna)** TRUE si la orden de disparo es UPDATE y la columna especificada ha sido actualizada; FALSE en otro caso.

# Ejemplo de UNION DE INSERT, DELETE Y UPDATE EN UN SOLO TRIGGER

```
CREATE OR REPLACE TRIGGER tdepart BEFORE DELETE OR INSERT OR UPDATE ON
    DEPART
FOR EACH ROW
declare
BEGIN
IF INSERTING THEN
DBMS_OUTPUT.PUT_LINE('Incidencia, nuevo departamento');
END IF;
IF DELETING THEN
DBMS_OUTPUT.PUT_LINE('Incidencia, borrado de departamento');
END IF;
IF UPDATING THEN
    DBMS_OUTPUT.PUT_LINE('Incidencia, actualizado de departamento');
END IF;
END;
/
```



# Ejemplo de UNION DE INSERT, DELETE Y UPDATE EN UN SOLO TRIGGER

- Comprobación:

```
Sql> insert into depart values (90,'prueba',1,7566,'Aaaaaaa',1000,10);
```

1 filas insertadas.

Incidencia, nuevo departamento

```
Sql> update depart set dnombre='PRUEBAS' where dept_no=90;
```

1 filas actualizadas.

Incidencia, actualizado de departamento

	DEPT_NO	DNOMBRE	NUMCE	DIREC	TDIR	PRESU	DEPDE
1	90	PRUEBAS	1	7566	A	1000	10
2	10	CONTABILIDAD	2	7782	F	-1233,8	30
3	20	INVESTIGACION	1	7566	P	12346,897	(null)
4	30	VENTAS	2	7698	P	30000	40
5	40	PRODUCCION	3	(null)	(null)	(null)	(null)

```
Sql> delete from depart where dept_no=90;
```

confirmadas.

1 filas eliminado

# Bibliografía

- <http://elbauldelprogramador.com/plsql-declaracion-de-variables/>
- <http://www.techonthenet.com/oracle>
- <http://devjoker.com/contenidos/catss/52/Procedimientos-almacenados-en-PLSQL.aspx>