

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO  
FACULTAD DE INGENIERÍA  
DEPARTAMENTO DE COMPUTACIÓN

# **Tema V**

## **Introducción a la programación con SQL**

Ing. Lucila Patricia  
Arellano Mendoza  
2018

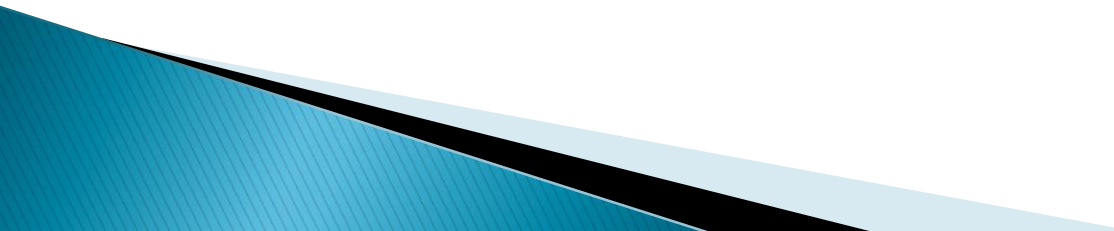


# Programación en ORACLE

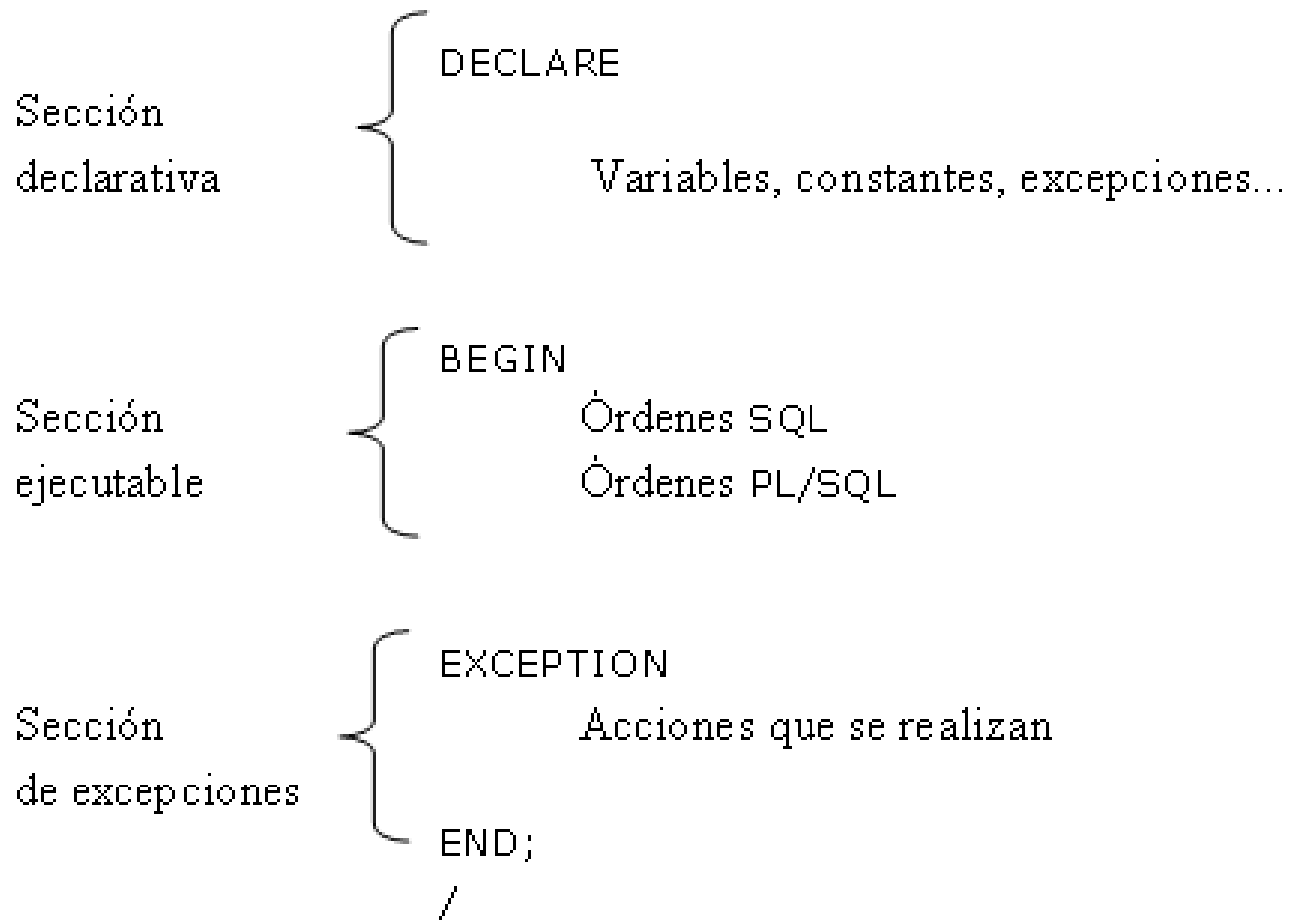
El lenguaje que se emplea para programar varía de un RDBMS a otro, el que utiliza Oracle se llama PL/SQL (Procedural Lenguaje/SQL)

# PL/SQL

Cada unidad pl/sql puede contener uno o más bloques que pueden o no estar anidados. Un bloque pl/sql puede ser anónimo o nominal, también es llamado subprograma ya sea función, procedimiento o disparador.



# Un bloque anónimo PL/SQL está compuesto de tres partes principales:



# La estructura general es:

**DECLARE**

Variables, constantes, cursores...

**BEGIN**

Órdenes SQL

Órdenes PL/SQL

**EXCEPTION**

Acciones a realizar

**END;**

/



Para ejecutar un bloque pl/sql hay que colocar al final la barra /

Para añadir comentarios al código es igual que en SQL

-- en línea

/\* párrafo \*/



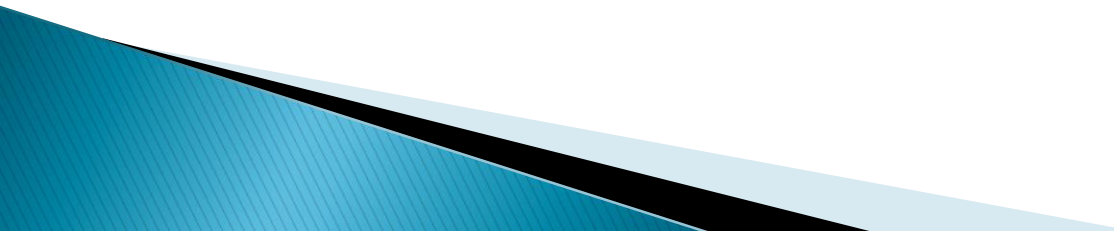
# Declaración de variables

- ▶ Las variables se caracterizan por su nombre, compuesto por letras, números y los caracteres \$, \_ ó # y su longitud puede ser máximo de 30 caracteres.
- ▶ Sintaxis:

identificador [CONSTANT] tipo de dato [NOT NULL] [:= | expresión];



# Tipos de datos

- ▶ Escalar. Son los mismos que las columnas de las tablas
  - ▶ Compuesto. Hacen referencia a un objeto existente
  - ▶ LOB. Datos no estructurados
- 



# Variables de tipo escalar

Las variables declaradas como NOT NULL siempre deben ser inicializadas. La inicialización puede hacerse utilizando

`:=`

o la palabra reservada  
`DEFAULT`

Si una variable no se inicializa contendrá el valor NULL.

Las constantes deben ser inicializadas.



# Asignación de valores a variables

Sintaxis:

identificador := valor;

Ejemplo:

vNombre:='Mariana Ruíz';

vClaveEmp:=234;



# Ejemplo:

DECLARE

vfecha DATE;

vDepNum NUMBER(2) NOT NULL := 10;

vCiudad VARCHAR(12) := 'Ciudad Real';

vKmMilla CONSTANT NUMBER(2,1) := 1.4;

# Comando SET

Se pueden realizar operaciones entre variables y ser almacenadas

Sintaxis:


SET campo = campo + vCampo ;

# Variables BOOLEANAS

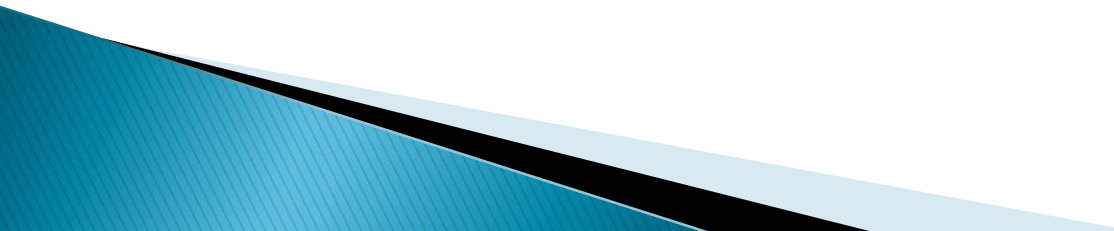
Las variable BOOLEANAS pueden tomar el valor TRUE, FALSE o NULL.

Las variables pueden combinarse mediante operadores lógicos (NOT, AND, OR).

Las expresiones pueden devolver valores BOOLEANOS utilizando operadores relacionales (<, <=...).



# Variables de tipo compuesto

- ▶ %TYPE. Permite hacer referencia a una columna de una tabla o a una variable definida con anterioridad.
  - ▶ %ROWTYPE. Hace referencia a estructuras completas de tablas o cursor
- 

# %TYPE

Sintaxis:

nomVariable {variable | tabla.columna}%TYPE;

Ejemplo:

vBalance NUMBER(5,2);

vBalanceMinimo vBalance%TYPE := 10;

vNombre empleado.nombre%TYPE;



# %ROWTYPE

Sintaxis:

```
nombreVariable {tabla | cursor} %ROWTYPE;
```

Ejemplo:

```
vArticulo articulo%ROWTYPE;
```





# Registros

Conjuntos de variables de diferente tipo que estan relacionados entre sí.

Sintaxis :

```
TYPE rRegistro IS RECORD(  
    Campo1 tipo1 [NOT NULL] [ :=expr1 ],  
    Campo2 tipo2 [NOT NULL] [ :=expr2 ],  
    Campon tipon [NOT NULL] [ :=exprn ]  
);
```

Ejemplo:

DECLARE

```
TYPE rCliente IS RECORD(  
    numCli NUMBER(4),  
    nomCli VARCHAR(30),  
    direcCli VARCHAR(50)  
);
```

vCliente rCliente;



# Órdenes SQL en PL/SQL

Las órdenes que se pueden ejecutar dentro de pl/sql son únicamente las del lenguaje de manipulación de datos, así como sentencias de control de flujo.

# SELECT

Sintaxis:

```
SELECT lista atributos  
INTO variable [, variable, variable...]  
FROM tabla  
WHERE condición;
```

Nota: Asegurar que sólo se devuelva una fila o tupla



Para presentar la salida de la ejecución de los bloques en pantalla se utiliza el paquete DBMS\_OUTPUT habilitando para ello el comando SET SERVEROUTPUT ON

## Sintáxis

```
DBMS_OUTPUT.PUT_LINE('Mensaje '||nombre);
```



Ejemplo:

```
SET SERVEROUTPUT ON;
```

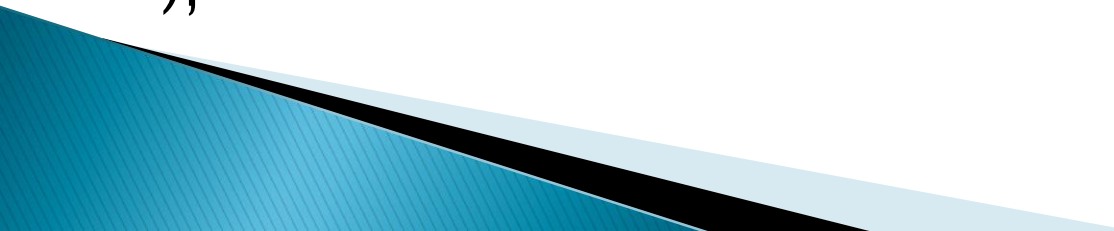
```
DBMS_OUTPUT.PUT_LINE('El nombre de usuario es : '|| vNombre);
```

A decorative graphic element in the bottom-left corner of the slide, consisting of overlapping blue and black geometric shapes.

Abra un archivo spool y realice los que se pide  
Ejemplo 9. Lenguaje procedural

Cree la siguiente tabla llamada area

```
CREATE TABLE area(  
claveArea NUMBER(5) NOT NULL,  
nomArea VARCHAR2(20),  
seccion CHAR(1),  
CONSTRAINT pkArea PRIMARY KEY (claveArea),  
CONSTRAINT ckTipoSec CHECK(seccion IN('A','B','C'))  
);
```



Inserte los siguientes valores:

CLAVEAREA	NOM AREA	SEC
-----	-----	-----
123	obras	A
234	sanidad	A
345	transporte	B
456	vialidades	B
567	salud	C



# IMPORTANTE

Activar la variable:

```
SET SERVEROUTPUT ON;
```

Vamos a crear un bloque anónimo que muestre los datos del área de la sección C

```
/* Bloque anónimo que muestra los datos del  
   área cuya sección es 'C' */
```

```
DECLARE
```

```
  vArea NUMBER(5);
```

```
  vNombre VARCHAR(20);
```

```
BEGIN
```

```
  SELECT claveArea, nomArea
```

```
  INTO vArea, vNombre
```

```
  FROM area
```

```
  WHERE seccion='C';
```

```
  DBMS_OUTPUT.PUT_LINE('area '||vArea);
```

```
  DBMS_OUTPUT.PUT_LINE('nombre '||vNombre);
```

```
END;
```

```
/
```

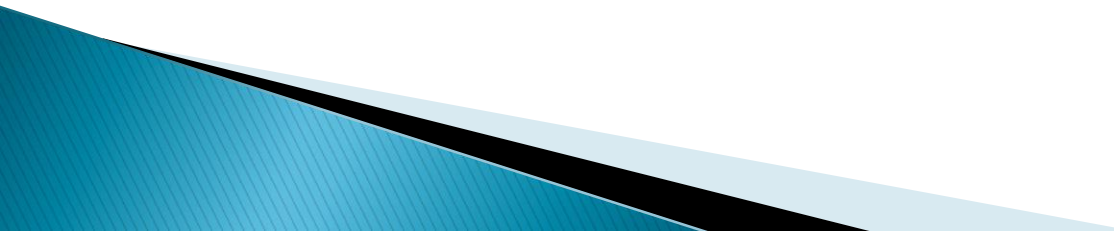


- ▶ Podemos guardar el bloque anónimo en un archivo para después volverlo a utilizar. Para ello utilizamos la siguiente sentencia:

```
SQL> SAVE [ruta]<nombreArchivo.sql>[replace]||[append]
```

Para ejecutarlo posteriormente realizamos

```
SQL> START <[ruta]><nombreArchivo.sql>
```



Para nuestro ejemplo

```
SQL> SAVE c:\temp\muestraArea.sql
```

Para ejecutarlo posteriormente realizaremos

```
SQL> START c:\temp\muestraArea.sql
```



# Variables de sustitución &, &&

Nos permiten solicitar datos por pantalla antes de ejecutar un bloque.

Estas variables tienen que ir antepuestas del & para que funcionen. El && se utiliza para no tener que solicitar nuevamente el dato.

Use comillas simples para fechas y cadenas de caracteres.

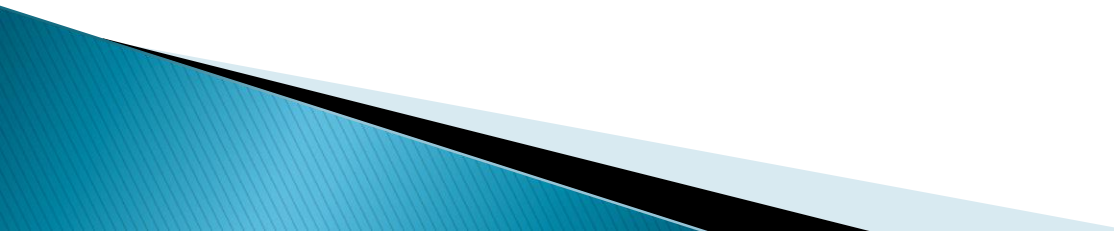
Utilice el comando VERIFY para visualizar el texto del comando antes y después de que la variable sea solicitada.

Sintaxis:

```
SET VERIFY {ON|OFF}
```

En el siguiente ejemplo se pide la clave del área y nos muestra su nombre.

```
DECLARE
    vNomArea area.nomArea%TYPE;
BEGIN
    SELECT nomArea INTO vNomArea
    FROM area
    WHERE claveArea= &claveArea;
    DBMS_OUTPUT.PUT_LINE (vNomArea);
END;
/
```



# Órdenes INSERT, UPDATE, DELETE

La sintaxis no varía:

INSERT INTO tabla VALUES( ... );

UPDATE tabla  
SET valor = expresión  
WHERE condición;

DELETE FROM tabla WHERE condición;



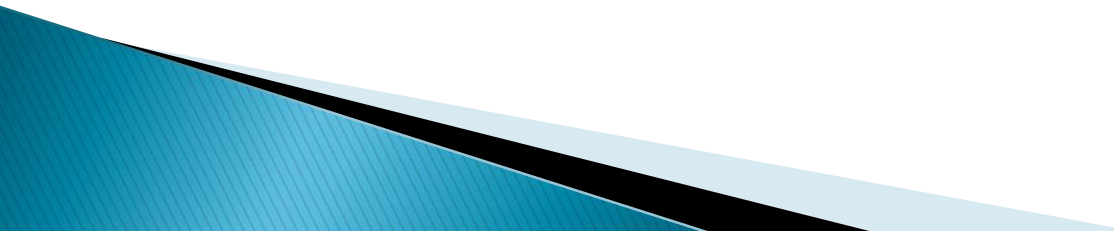
# Órdenes PL/SQL

Dentro de PL/SQL se puede hacer uso de funciones numéricas, de carácter, de fecha y conversión de tipos.

Los operadores en PL/SQL son los mismos que en SQL: aritméticos, lógicos, concatenación y paréntesis.



También se utilizan estructuras de control, éstas permiten elegir la forma en la que se van a ejecutar las instrucciones dentro del programa.

- ▶ Condicionales
  - ▶ Repetición (ciclos)
- 

# Estructuras condicionales

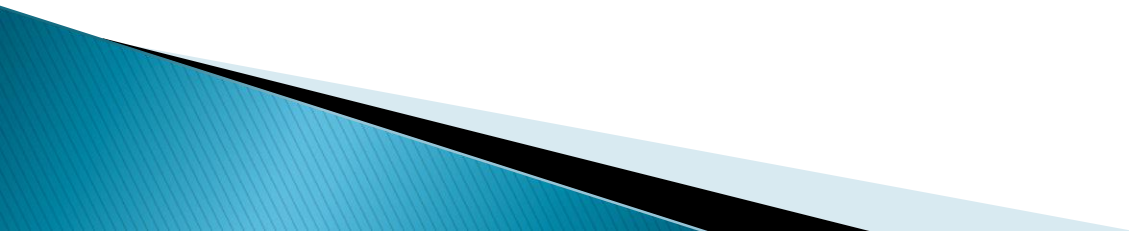
Sentencia IF

Sintaxis:

IF- THEN-END IF;

IF-THEN-ELSE-END IF;

IF-THEN-ELSIF-END IF;



# Sentencia CASE

## Sintaxis

### CASE selector

WHEN expresion1 THEN instrucciones

WHEN expresión2 THEN instrucciones

....

WHEN expresionN THEN instrucciones

[ELSE instrucciones];

END CASE;



# Estructuras de repetición

## Sentencia LOOP

Sintaxis:

LOOP

    instrucciones

    [EXIT WHEN condición]

END LOOP;



# Sentencia FOR

Sintaxis:

```
FOR índice IN [REVERSE] min..max LOOP
    instrucciones;
    ...
END LOOP;
```

# Sentencia WHILE

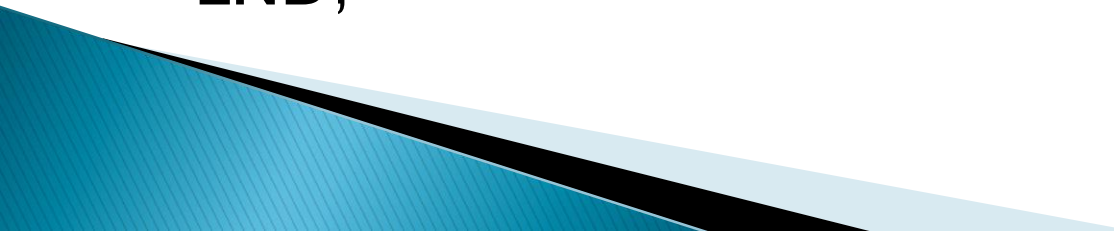
Sintaxis:

```
WHILE condición LOOP
    instrucciones;
    ...
END LOOP;
```

# Excepciones en pl/sql

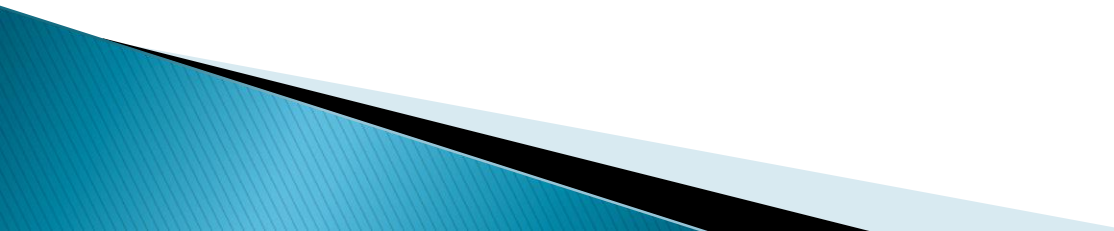
Una excepción en pl/sql es una advertencia o condición de error. Cuando se produce un error PL/SQL levanta una excepción y pasa el control a la sección de excepción correspondiente al bloque PL/SQL. Su estructura es la siguiente:

```
DECLARE  
  -- Declaraciones  
BEGIN  
  -- Ejecución  
EXCEPTION  
  -- Excepción  
END;
```

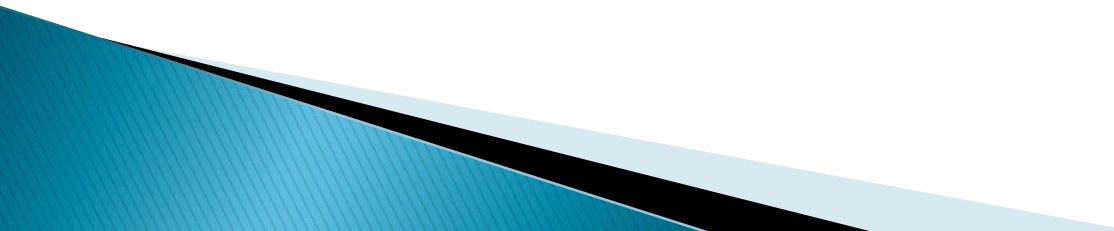


# Excepciones predefinidas

Son aquellas que se disparan automáticamente al producirse determinados errores. Las más comunes son:

- ❖ **TOO\_MANY\_ROWS:** Se produce cuando `SELECT ... INTO` devuelve más de una fila.
  - ❖ **NO\_DATA\_FOUND:** se produce cuando `SELECT ... INTO` no devuelve ninguna fila.
  - ❖ **PROGRAM\_ERROR:** se produce cuando hay un problema interno en la ejecución del programa.
- 



- ❖ **VALUE\_ERROR**: se produce cuando hay un error aritmético o de conversión.
  - ❖ **ZERO\_DIVIDE**: se produce cuando hay una división entre 0.
  - ❖ **INVALID\_NUMBER**: se produce cuando se intenta convertir una cadena a un valor numérico.
- 

# Ejemplo de estructura del bloque

**DECLARE**

-- Declaraciones

**BEGIN**

-- Ejecución

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

-- Cuando ocurre una excepción DATO NO ENCONTRADO

**WHEN ZERO\_DIVIDE THEN**

-- Cuando ocurre una excepción DIVISION ENTRE CERO

**WHEN OTHERS THEN**

-- Cuando ocurre una excepción de un tipo no tratado

-- en los bloques anteriores

**END;**

/

# Excepciones definidas por el usuario

- ▶ Son aquellas que crea el usuario. Para ello se requieren tres pasos:
  1. Definición: se realiza en la zona de DECLARE con el siguiente formato:  
nombre\_excepción EXCEPTION
  2. Disparar o levantar la excepción mediante la orden raise:  
RAISE ... ;
  3. Tratar la excepción en el apartado  
EXCEPTION  
WHEN ... THEN ;

Ejemplo:

DECLARE

...  
exImporteMal EXCEPTION;

...  
BEGIN

...  
IF precio NOT BETWEEN mínimo AND máximo  
THEN  
    RAISE exImporteMal;  
END IF;

...  
EXCEPTION  
WHEN exImporteMal  
THEN  
    DBMS\_OUTPUT.PUT\_LINE('Importe incorrecto');

...  
END;



# RAISE\_APPLICATION\_ERROR

- ▶ En PL/SQL se incluye un procedimiento llamado RAISE\_APPLICATION\_ERROR que nos sirve para levantar errores y definir mensajes de error. Su formato es el siguiente:

`RAISE_APPLICATION_ERROR(númeroError,mensajeError);`

El número de error esta comprendido entre -20000 y -20999 y el mensaje es una cadena de caracteres de hasta 512 bytes.

# RAISE\_APPLICATION\_ERROR

Se utiliza en dos lugares distintos:

- Sección Ejecutable
- Sección de Excepciones

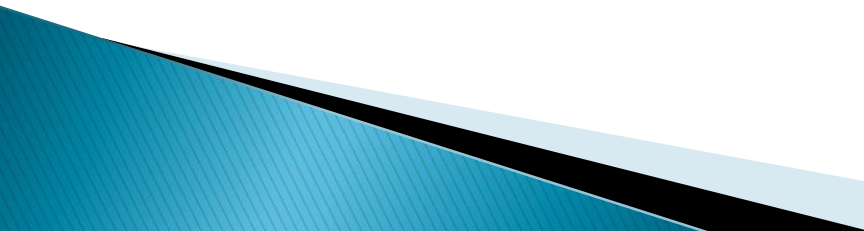
# Ejemplos:

## Sección ejecutable

```
BEGIN
  IF SQL % NOTFOUND THEN
    RAISE_APPLICATION_ERROR (-20001, 'No existen registros' );
  END IF;
END;
```

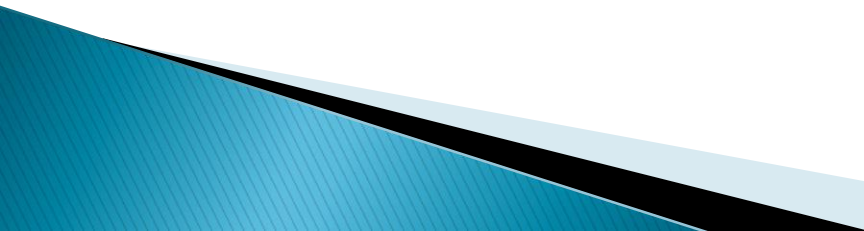
## Sección de excepciones

```
EXCEPTION
WHEN NO_DATA_FOUND THEN
  RAISE_APPLICATION_ERROR (-20002, 'Datos no encontrados' );
END;
```



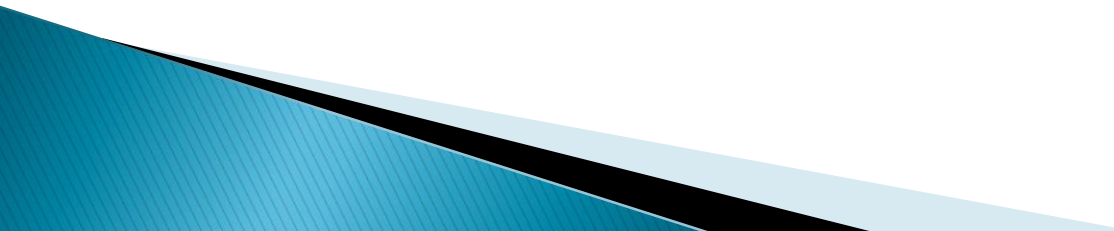
# Ejemplo de RAISE\_APPLICATION\_ERROR

```
BEGIN
  SELECT horas INTO vHorasActuales
  FROM empleado
  WHERE idEmpleado=vEmpleado;
  IF vHorasActuales IS NULL
  THEN
    RAISE_APPLICATION_ERROR(-20010,'No tiene horas');
  ELSE
    UPDATE empleado
    SET horas=vHorasActuales + subirHoras
    WHERE idEmpleado=vEmpleado;
  END IF;
END;
```





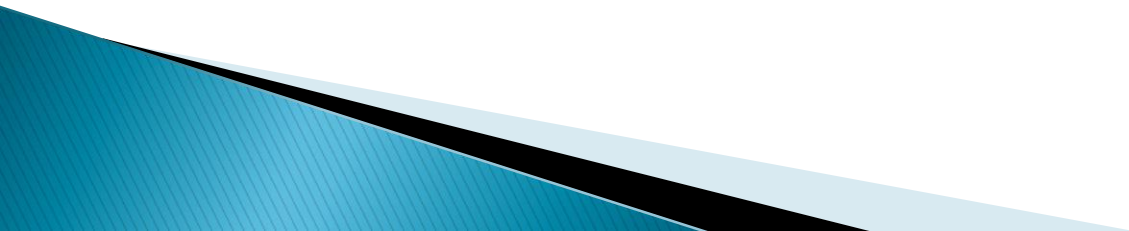
# Reglas de Alcance

- ▶ Una excepción es válida dentro del bloque o programa donde ha sido declarada.
  - ▶ Las excepciones predefinidas son siempre válidas.
  - ▶ Una excepción declarada en un bloque es local a ese bloque y global a todos los sub bloques que comprende.
- 

# Procedimientos almacenados

Es un conjunto de comandos SQL que pueden ser compilados y almacenados en el servidor.

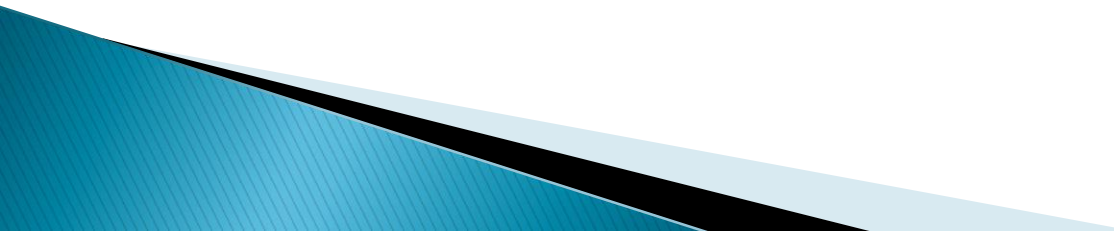
Los procedimientos son también llamados subprogramas.



# Procedimiento almacenado

## Sintaxis

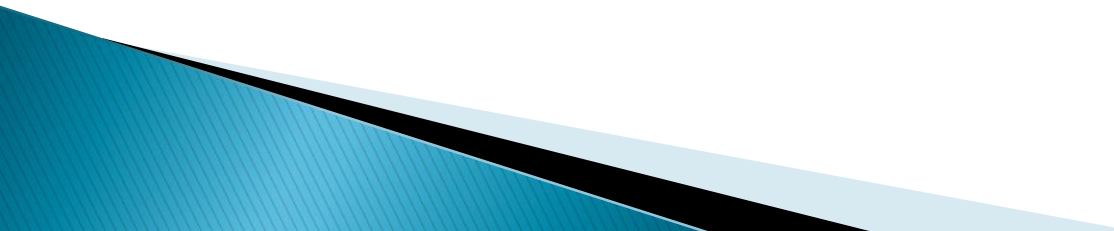
```
CREATE [OR REPLACE] PROCEDURE nombreProcedimiento  
(  
    argumento [IN|OUT|IN OUT] <tipo> [,  
    argumento [IN|OUT|IN OUT] <tipo>,...]  
)  
AS  
BEGIN  
    <código del procedimiento>  
[EXCEPTION]  
END nombreProcedimiento;  
/
```



Los parámetros de un procedimiento pueden tener valores predeterminados que se declaran mediante la siguiente sintaxis.

nombreArgumento [modo] <tipo> {:= | DEFAULT} valor inicial

# Ejecución

- ▶ Un procedimiento puede ejecutarse con el siguiente comando:
  - ▶ EXEC <nombreProcedimiento>(parámetros);
- 

# Para borrar un procedimiento:

`DROP PROCEDURE <nombreProcedimiento>;`

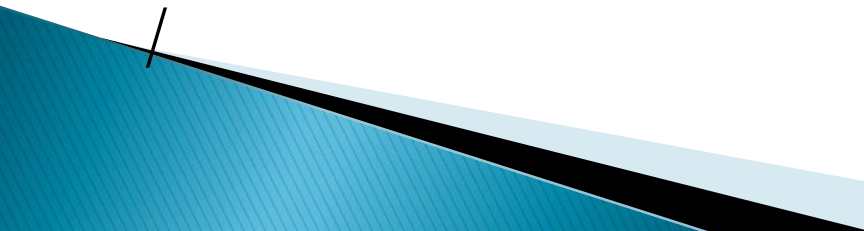
Si el procedimiento tiene errores de compilación los podemos observar ejecutando el comando:

`SQL>SHOW ERRORS`



Ejemplo : Procedimiento para dar de alta un área, realice lo siguiente:

```
CREATE OR REPLACE PROCEDURE spAltaArea(  
    vArea IN NUMBER,  
    vNom IN VARCHAR2,  
    vSec IN CHAR)  
AS  
BEGIN  
    INSERT INTO area VALUES(vArea,vNom,vSec);  
    COMMIT;  
    DBMS_OUTPUT.PUT_LINE('Area creada: '||vNom);  
END spAltaArea;
```



Para ejecutarlo

```
EXEC spAltaArea(235,'Mecatrónica','A');
```

Obteniéndose :

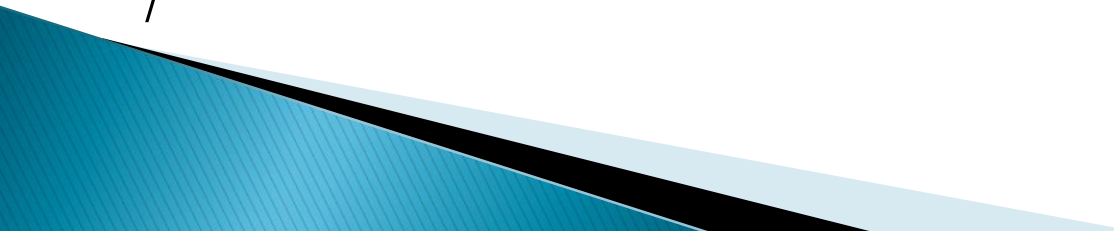
Área creada: Mecatrónica





Otra forma de crear el procedimiento es utilizando %TYPE

```
CREATE OR REPLACE PROCEDURE spAltaNuevaArea(  
    vArea area.claveArea%TYPE,  
    vNom area.nomArea%TYPE,  
    vSec area.seccion%TYPE  
)  
AS  
BEGIN  
    --Inserta una nueva fila en la tabla área  
    INSERT INTO area (claveArea,nomArea,seccion)  
    VALUES (vArea,vNom,vSec);  
    COMMIT;  
END spAltaNuevaArea;  
/
```



Para proporcionar los parámetros al procedimiento anterior realizamos

```
BEGIN
```

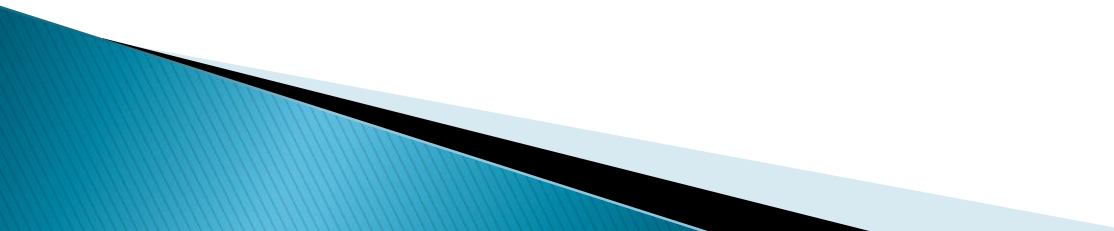
```
  spAltaNuevaArea (132, 'Transferencia', 'A');
```

```
END;
```

```
/
```

o bien

```
EXEC spAltaNuevaArea(232,'Contratación','B');
```



Ejemplo: Procedimiento para dar de baja un área

```
CREATE OR REPLACE PROCEDURE spBajaArea
(vArea IN NUMBER)
AS
BEGIN
    DELETE FROM area
    WHERE claveArea=vArea;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Baja del Área');
END spBajaArea;
```

/



Para ejecutarlo:

```
EXEC spBajaArea(1 23);
```

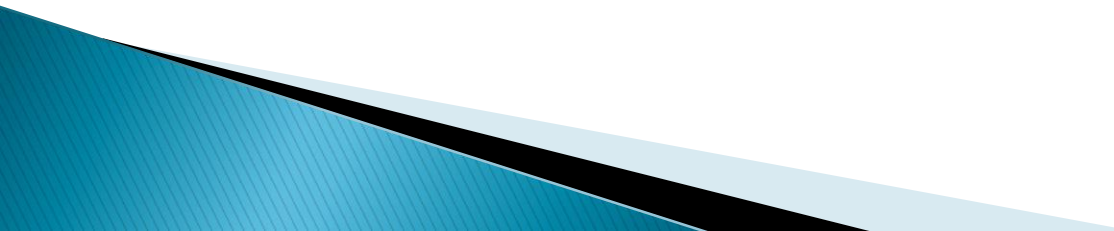
Obteniéndose:

Baja del Área



## Ejemplo: Modificación del nombre de un área

```
CREATE OR REPLACE PROCEDURE spCambiaArea
(vArea IN NUMBER, vNom IN VARCHAR2)
AS
BEGIN
    UPDATE area SET nomArea=vNom
    WHERE claveArea=vArea;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Área modificada: '||vNom);
END spCambiaArea;
/
```



# Ejecutando el procedimiento

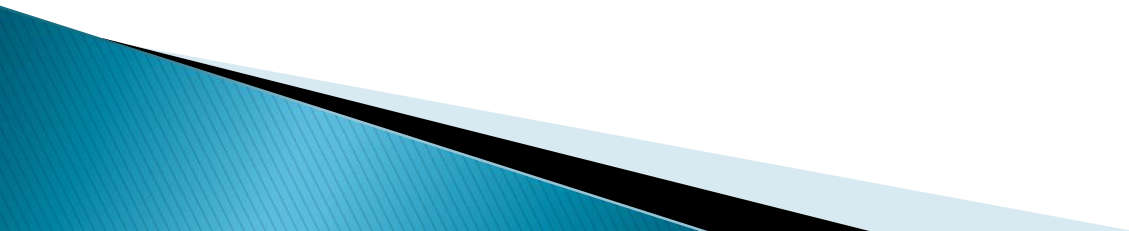
```
EXEC spCambiaArea(234,'Ciencias');
```

Obteniéndose:


Área modificada: Ciencias

Ejemplo:

En el caso, de que se necesite cambiar los datos de un área, de tal forma que si no existe se cree un nuevo registro. El código será el siguiente:



```
CREATE OR REPLACE PROCEDURE spActualizaArea(  
    vArea area.claveArea%TYPE,  
    vNom area.nomArea%TYPE,  
    vSec area.seccion%TYPE  
)  
AS  
    vBuscaArea area.claveArea%TYPE;  
    vNumArea area.claveArea%TYPE;  
BEGIN  
    SELECT claveArea INTO vBuscaArea  
    FROM area  
    WHERE claveArea=vArea;  
  
    IF (vBuscaArea IS NOT NULL) THEN  
        DBMS_OUTPUT.PUT_LINE('Área existente '|| vArea);  
        UPDATE area SET nomArea=vNom, seccion=vSec  
        WHERE claveArea=vArea;  
    END IF;
```





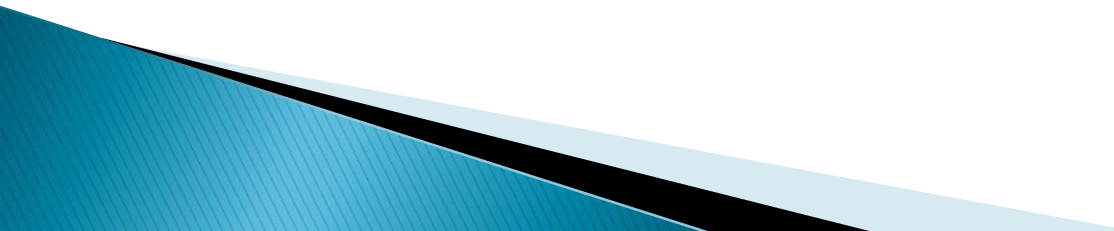
```
EXCEPTION WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Área no encontrada' ||vArea);

    SELECT MAX(claveArea)
    INTO vNumArea
    FROM area;

    INSERT INTO area
    VALUES(vNumArea+1 , vNom, vSec);

    COMMIT;

END spActualizaArea;
/
```



Ejecutamos el procedimiento

```
EXEC spActualizaArea(234,'Civil','C');
```

```
EXEC spActualizaArea(121,'Hidraulica','B');
```

Muestre los datos para observar el resultado

```
SELECT * FROM area;
```



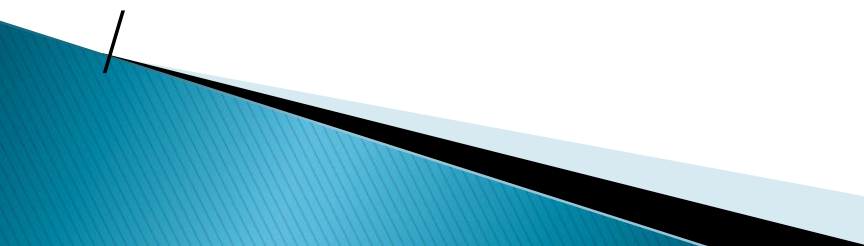
Cree la siguiente tabla e inserte los datos utilizando un procedimiento de inserción

```
CREATE TABLE salario (  
    empleo VARCHAR2(20) PRIMARY KEY,  
    sueldo NUMBER(7,2)  
);
```

EMPLEO	SUELDO
Capturista	5000
Secretaria	3000
Administrador	6000
Gerente	10000

Este procedimiento actualiza el valor del sueldo de los Capturistas según el factor proporcionado.

```
CREATE OR REPLACE PROCEDURE spAjusteSalario
(vFactor IN NUMBER)
AS
BEGIN
    UPDATE salario
    SET sueldo=sueldo*vFactor
    WHERE empleo='Capturista';
    COMMIT;
END spAjusteSalario;
```



Ejecutelo utilizando

```
EXEC spAjusteSalario(1.4);
```

Cierre su archivo spool

Ejemplo: Estructuras condicionales y de repetición  
Abra un nuevo archivo spool y realice lo siguiente.

Cree la tabla Cargo, inserte los datos y confírmelos


tipoCargo	descCargo	costo	fechaRealiz
1	Servicio	4000.00	7/09/17
2	Actualizacion	1500.00	7/09/17
3	Configuracion	3549.95	7/09/17
4	Planeacion	10500.90	7/09/17
5	Disenio	15899.90	7/09/17

Incremente el costo de los cargos, según el tipo de cargo de la siguiente forma: para los cargos 4 y 5 en 10% y los restantes en 5%, actualice la fecha. Muestre los resultados obtenidos.

```
CREATE OR REPLACE PROCEDURE spActualizaCosto  
AS
```

```
    vCargoActual NUMBER;  
    vNumTipoCargo NUMBER;
```

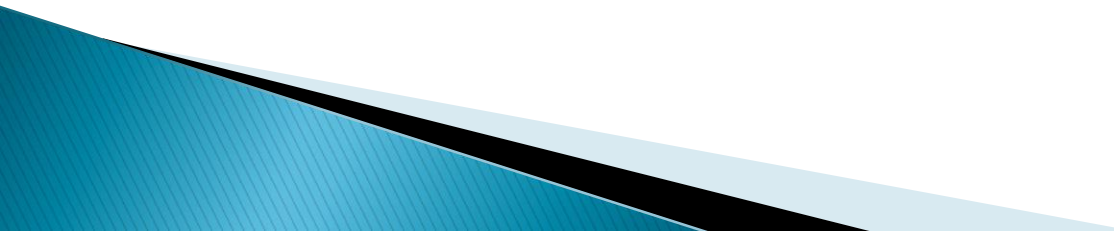
```
BEGIN  
    SELECT COUNT(tipoCargo)  
    INTO vNumTipoCargo  
    FROM cargo;
```



```
FOR indice IN 1.. vNumTipoCargo LOOP
  SELECT costo
  INTO vCargoActual
  FROM cargo
  WHERE tipoCargo = indice;

  IF (indice = 4 OR indice = 5) THEN
    UPDATE cargo
    SET costo=vCargoActual*1.1, fechaRealiz = SYSDATE
    WHERE tipoCargo = indice;
  ELSE
    UPDATE cargo
    SET costo=vCargoActual*1.05, fechaRealiz = SYSDATE
    WHERE tipoCargo = indice;
  END IF;

END LOOP;
```





```
DBMS_OUTPUT.PUT_LINE('Cargos Actualizados');  
COMMIT;  
END spActualizaCosto;  
/
```

Ejecutamos el procedimiento y verificamos el resultado

```
EXEC spActualizaCosto;
```

```
SELECT * FROM cargo;
```

Cierre su archivo spool