

# SEMINARIO

**PL/SQL**

**Grado en  
Ingeniería  
Informática**



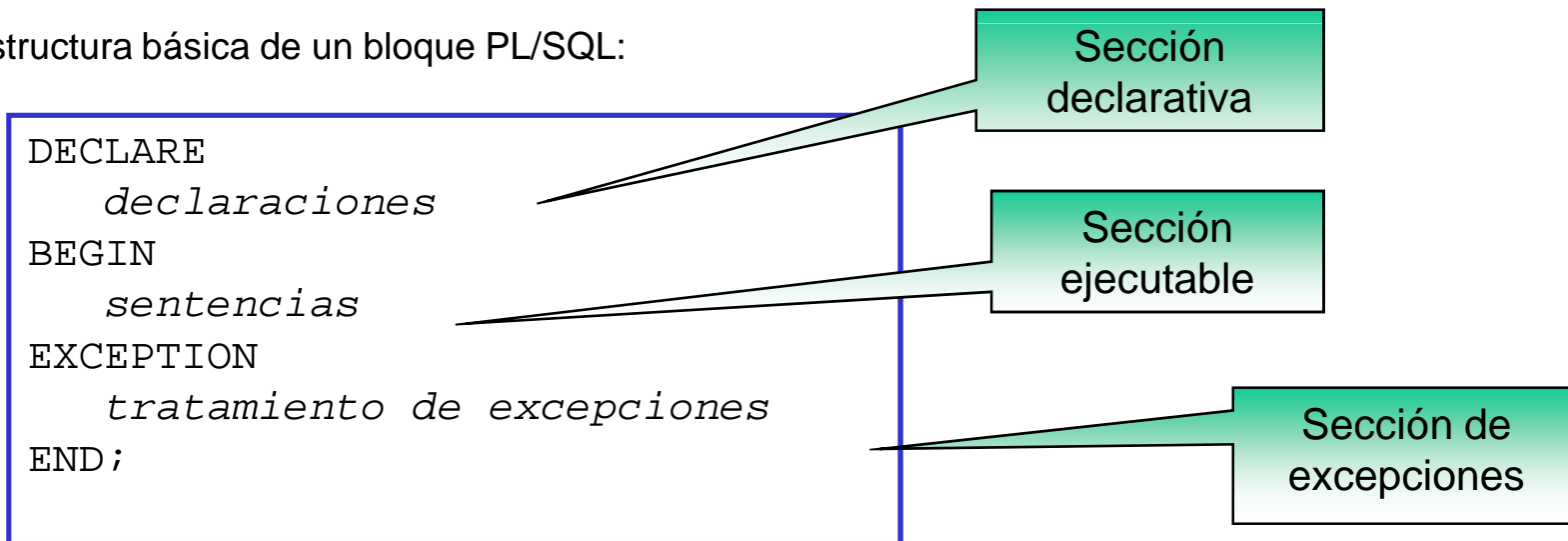
**Bases de  
Datos**

Departamento de Tecnologías de la Información  
Universidad de Huelva

## Índice

1. Introducción
2. Ejecución de PL/SQL
3. Variables, constantes y tipos
4. Estructuras de control de flujo
5. SELECT ... INTO
6. Cursores
7. Gestión de Excepciones
8. Procedimientos y Funciones
9. Ejemplo Resumen

- ❑ PL/SQL combina el uso de sentencias SQL y el flujo de control de un lenguaje procedural
- ❑ Lenguaje completo: sentencias para declarar y manipular variables, control de flujo de proceso, definición de procedimientos y funciones, gestión de excepciones
- ❑ Lenguaje estructurado en bloques: las unidades básicas (procedimientos, funciones y bloques anónimos) son bloques lógicos, que pueden contener cualquier número de bloques anidados
- ❑ Estructura básica de un bloque PL/SQL:



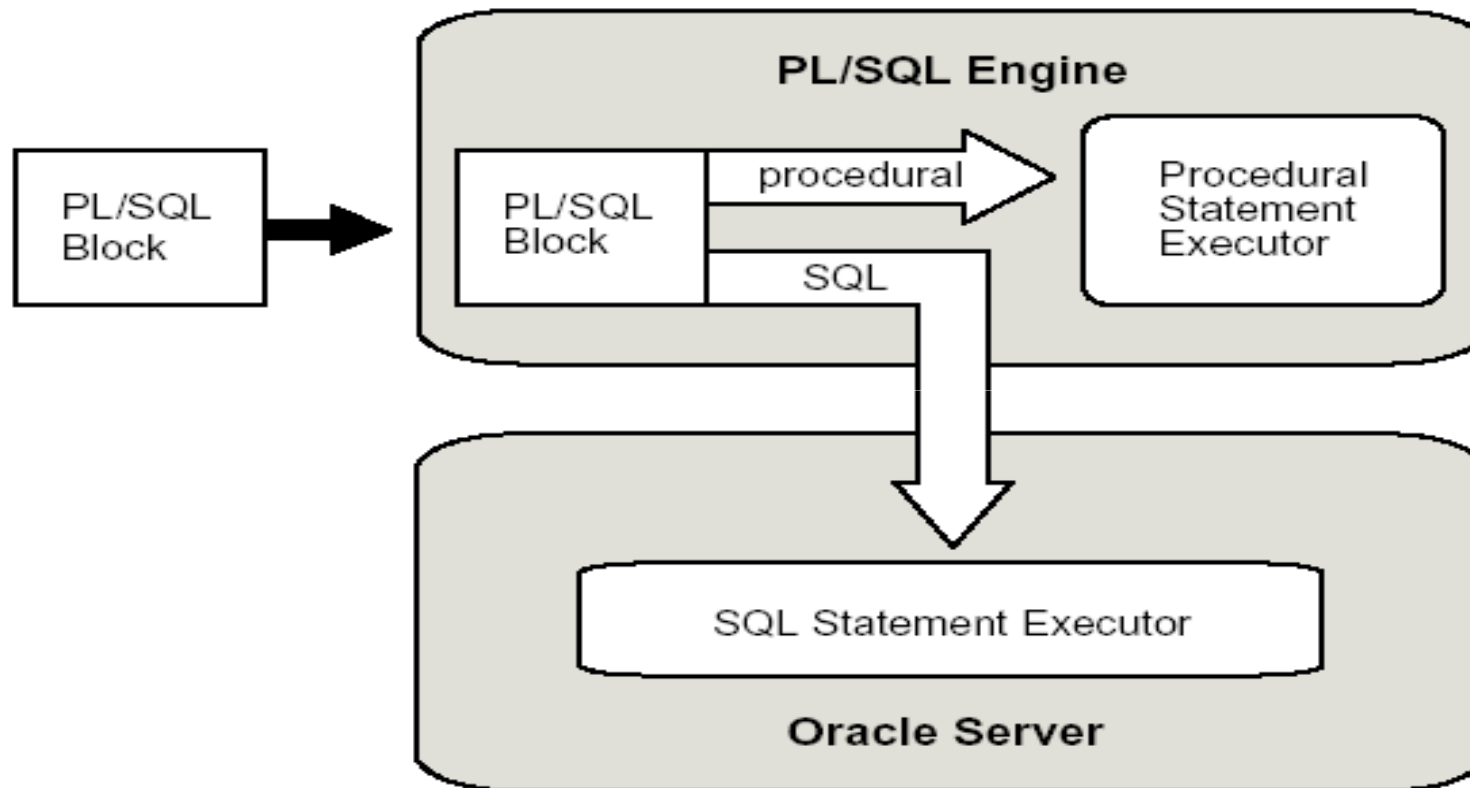
## 2. Ejecución de PL/SQL

---

- ❑ El motor de compilación y ejecución de código PL/SQL puede estar instalado en un servidor de base de datos Oracle o en una herramienta de aplicaciones Oracle (*Forms, Reports*)
- ❑ El motor PL/SQL identifica en tiempo de ejecución qué parte es propiamente procedural y qué parte son sentencias SQL que envía al servidor Oracle para su ejecución
- ❑ Los procedimientos y funciones PL/SQL pueden ser compilados y guardados en la base de datos permanentemente (**subprogramas almacenados**), listos para ser llamados por los usuarios y aplicaciones
- ❑ Los **subprogramas almacenados** pueden ser llamados desde un disparador de la base de datos, desde otro subprograma, desde código escrito en otros lenguajes o interactivamente desde herramientas como SQL Developer

Ejemplo de llamada desde SQL Developer

```
CALL porcentaje_aprobados('Bases de Datos', 2002);
```



### 3. Variables, Constantes y Tipos

---

- ☐ Las variables deben declararse antes de ser utilizadas
- ☐ Pueden ser de cualquier tipo existente en SQL, así como BOOLEAN
- ☐ Las constantes se declaran anteponiendo CONSTANT antes de su tipo
- ☐ Por defecto, las variables son inicializadas a NULL

#### Sintaxis

```
nombre_variable [CONSTANT] tipo [NOT NULL] [:= expresion];
```

#### Ejemplos

```
fecha_nacimiento DATE;  
contador NUMBER(7,0) := 0;  
categoria VARCHAR2(80) := 'Vendedor';  
pi CONSTANT NUMBER := 3.14159;  
radio NUMBER := 5;  
area NUMBER := pi * radio**2;
```

### 3. Variables, Constantes y Tipos

---

- ❑ El lenguaje PL/SQL permite declarar algunos [tipos compuestos](#)
- ❑ El tipo VARRAY permite declarar vectores (colección ordenada de elementos del mismo tipo)

```
TYPE nombre IS VARRAY (limite_tamaño) OF tipo_elemento [NOT NULL];
```

- ❑ El tipo RECORD permite declarar registros (composición de variables de tipos diferentes en un mismo grupo lógico)

```
TYPE nombre IS RECORD (declaracion_campo [, declaracion_campo] ...);
```

donde [declaracion\\_campo](#):

```
nombre_campo nombre_tipo [ [NOT NULL] {:= | DEFAULT} expresion ]
```

- ❑ Se pueden definir subtipos basados en los tipos base o en otros subtipos

```
SUBTYPE nombre IS tipo [NOT NULL];
```

### 3. Variables, Constantes y Tipos

---

- ❑ Con %TYPE y %ROWTYPE se pueden referenciar los tipos de los atributos o tuplas de tablas existentes, respectivamente

```
TYPE alumnoReg IS RECORD (  
    codAlumno ALUMNO.nAl%TYPE,  
    comentarios VARCHAR2(90) );
```

```
SUBTYPE profesorReg IS PROFESOR%ROWTYPE;
```

- ❑ Se pueden asignar valores a las variables de las siguientes maneras:
  - Usando el operador de asignación :=
  - Usando SELECT ... INTO (la consulta debe devolver una única tupla)
  - Mediante el paso de parámetros en llamadas a procedimientos o funciones

#### Ejemplos

```
contador := contador + 1;  
SELECT fechaNac INTO fecha_nacimiento FROM ALUMNO  
WHERE nAl = 26;
```



## 4. Estructuras de Control de Flujo

---

### ❑ IF-THEN-ELSE

```
IF condicion THEN  
    secuencia de sentencias  
ELSIF condicion THEN  
    secuencia de sentencias  
    ...  
ELSE  
    secuencia de sentencias  
END IF;
```

### ❑ CASE

```
CASE selector  
    WHEN expresion THEN secuencia de sentencias  
    WHEN expresion THEN secuencia de sentencias  
    ...  
    WHEN expresion THEN secuencia de sentencias  
    [ELSE secuencia de sentencias]  
END CASE;
```

## 4. Estructuras de Control de Flujo

---

☐ Bucles WHILE

```
WHILE condicion LOOP  
    secuencia de sentencias  
END LOOP;
```

☐ Bucles FOR-LOOP

```
FOR indice IN [REVERSE] limite_inferior .. limite_superior LOOP  
    secuencia de sentencias  
END LOOP;
```

☐ Bucles LOOP continuos

```
LOOP  
    secuencia de sentencias  
END LOOP;
```

☐ Interrupción de bucles

```
EXIT [WHEN condicion]
```

- ❑ Extrae datos de la base de datos y los almacena en variables PL/SQL

Sintaxis

```
SELECT lista_selección INTO lista_variables  
FROM referencia_tabla  
[WHERE cláusula_where];
```

- Debe haber el **mismo número** de elementos de selección que de variables
- Cada variable debe ser **compatible** con su elemento asociado
- La instrucción SELECT ... INTO NO puede devolver más de una fila, pero tampoco ninguna

### Ejemplo

```
DECLARE
    r_asignatura ASIGNATURA%ROWTYPE;
    nomProf      PROFESOR.nombre%TYPE;
    desProf      PROFESOR.despacho%TYPE;
BEGIN
    ...
    SELECT * INTO r_asignatura
    FROM ASIGNATURA
    WHERE idAsig = 'A004';
    ...
    SELECT nombre, despacho INTO nomProf, desProf
    FROM PROFESOR
    WHERE nPr = 11;
    ...
END;
```

- ☐ Con un cursor se especifica un conjunto de tuplas (resultado de una consulta) que se pretenden procesar *de una en una*
- ☐ Puede declararse un cursor en la parte declarativa de cualquier bloque o subprograma

### Sintaxis


```
CURSOR nombre_cursor [ (lista_parametros) ] IS sentencia_select;
```

- ☐ Si se declaran parámetros en la definición del cursor, serán usados en la sentencia SELECT del cursor
- ☐ El uso de los cursores se realiza abriéndolos (OPEN), accediendo secuencialmente a sus tuplas (FETCH) y cerrándolos (CLOSE)

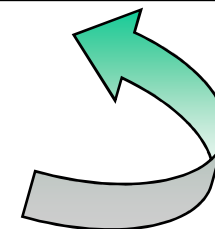
### Ejemplo de funcionamiento

```
CURSOR c_alumno IS SELECT nAl, nombre FROM ALUMNO WHERE lugar <> 'HUELVA';
```

c\_alumno



|     |                          |
|-----|--------------------------|
| 112 | Francisco Gallego Macías |
| 088 | Teresa Díaz Camacho      |
| 220 | Beatriz Rico Vázquez     |
| 111 | Antonio Resines Pérez    |
| 175 | Eva García Gil           |
| 149 | Pablo Gómez Ruíz         |



### Sintaxis

```
OPEN nombre_cursor [(lista_parametros)];
```

```
FETCH nombre_cursor INTO variables;
```

```
CLOSE nombre_cursor;
```

### Ejemplo

```
DECLARE
    CURSOR c_notas (al CHAR(3), curso NUMBER(4))
        IS SELECT idAsig, feb_jun, sep, dic FROM MATRICULA
           WHERE alum = al AND año = curso;
BEGIN
    ...
    OPEN c_notas(112, 2002);
    ...
END;
```

❑ Atributos que poseen todos los cursores:

- %FOUND: TRUE si el último FETCH devolvió una tupla
- %NOTFOUND: TRUE si el último FETCH no devolvió una tupla
- %ISOPEN: TRUE o FALSE según el cursor esté abierto o no
- %ROWCOUNT: número de tuplas accedidas (con FETCH) hasta ese momento

Ejemplo: procesamiento de 20 filas (como máximo) de la tabla ALUMNO

```
DECLARE
  CURSOR c_alumno IS SELECT * FROM ALUMNO WHERE lugar <> 'HUELVA';
  r_alumno ALUMNO%ROWTYPE;
BEGIN
  OPEN c_alumno;
  FETCH c_alumno INTO r_alumno;
  WHILE c_alumno%FOUND and c_alumno%ROWCOUNT <= 20 LOOP
    sentencias de procesamiento de r_alumno
    FETCH c_alumno INTO r_alumno;
  END LOOP;
  CLOSE c_alumno;
END;
```



- ❑ Se puede simplificar el uso de cursores mediante el [bucle FOR específico para cursores](#):

```
FOR registro IN cursor [(lista_parametros)] LOOP  
    secuencia de sentencias  
END LOOP;
```

- ❑ Se declara [implícitamente](#) la variable *registro*
- ❑ También [implícitamente](#) se ejecuta un **OPEN** antes de entrar por primera vez en el bucle y un **FETCH** al comienzo de cada iteración
- ❑ Antes de iniciar cada iteración, se comprueba si se ha alcanzado el final de las tuplas. De ser así, se [realiza un CLOSE](#) del cursor y se ejecuta la sentencia que sigue al **END LOOP**

- ❑ Ejemplo, supongamos el procesamiento de los alumnos nacidos fuera de Huelva:

```
DECLARE
  CURSOR c_alumno IS SELECT * FROM ALUMNO WHERE lugar <> 'Huelva';
BEGIN
  FOR r_alumno IN c_alumno LOOP
    sentencias
  END LOOP;
END;
```

- ❑ Es incluso posible no declarar tampoco el cursor, y especificar solamente la sentencia SELECT en el bucle FOR:

```
BEGIN
  FOR r_alumno IN (SELECT dni, nombre, ordenador
                   FROM ALUMNO ORDER BY nAl) LOOP
    sentencias
  END LOOP;
END;
```

- ☐ PL/SQL implementa los mecanismos de tratamiento de errores mediante el gestor de excepciones
- ☐ Una excepción es un **error** o **evento** durante la ejecución de un bloque
- ☐ Se pueden asociar excepciones a los errores de Oracle o a errores definidos por el usuario (programador)
- ☐ Cuando se produce un error, se genera una excepción y el control pasa al gestor de excepciones
- ☐ Las excepciones definidas por el sistema se '**disparan**' automáticamente, pero las definidas por el usuario se deben disparar explícitamente (mediante el comando RAISE) y declararse previamente (con el tipo EXCEPTION)
- ☐ En la sección EXCEPTION del bloque PL/SQL deben definirse las sentencias para el tratamiento de cada excepción

### Declaración de excepciones

- ☐ Las excepciones se declaran en la sección declarativa de un bloque
- ☐ Ejemplo:

```
DECLARE  
    e_no_existe_asignatura EXCEPTION;
```

### Tratamiento de las excepciones

- ❑ Cuando se produce un error asociado a una excepción, se genera dicha excepción y el control pasa a la sección EXCEPTION, donde es tratada

Sintaxis:

#### EXCEPTION

```
WHEN nombre_excepcion_1 THEN
    sentencias_tratamiento_e1;
WHEN nombre_excepcion_2 OR nombre_excepcion_3 THEN
    sentencias_tratamiento_e2_y_e3;
...
WHEN OTHERS THEN
    Este bloque de sentencias se ejecutará para cualquier otro error
    sentencias_tratamiento_otro_error;
```

Ejemplo: Control del número de ordenadores por aula (i.e.: máximo 25 ordenadores por aula)

```
DECLARE
    e_limite_ordenadores EXCEPTION; -- Declaración de la excepción
    v_numero_ordenadores NUMBER(2); -- Número de ordenadores en un aula
    v_maximo_ordenadores CONSTANT NUMBER(2):=25;
BEGIN
    /* Se calcula el número de ordenadores que hay en un aula */
    SELECT COUNT(*) INTO v_numero_ordenadores FROM ORDENADOR
    WHERE lugar = p_aula; -- p_aula es un parámetro

    /* Comprueba si se ha superado el número máximo */
    IF v_numero_ordenadores > v_maximo_ordenadores THEN
        RAISE e_limite_ordenadores;
    END IF;
    ...
END;
```

Ejemplo: Control del número de ordenadores por aula (sigue)

```
...  
EXCEPTION  
WHEN e_limite_ordenadores THEN  
    INSERT INTO tabla_de_control(error) VALUES ('El aula ' || p_aula ||  
        'tiene ' || v_numero_ordenadores || ' ordenadores');  
WHEN OTHERS THEN  
    INSERT INTO tabla_de_control(error) VALUES ('Ha ocurrido un error  
        desconocido');  
END;
```

- ☐ Se puede utilizar la función `RAISE_APPLICATION_ERROR` para crear mensajes de error propios
- ☐ Los errores definidos por el usuario se pasan fuera del bloque, al entorno que realizó la llamada
- ☐ Se pueden poner en el bloque de código y en el bloque de excepciones

Sintaxis:

```
RAISE_APPLICATION_ERROR (número de error, mensaje de error);
```

- *número de error* es un valor comprendido entre -20000 y -20999
- *mensaje de error* es el texto asociado al error

Ejemplo:

```
BEGIN
    ...
    IF v_numero_ordenadores > v_maximo_ordenadores THEN
        RAISE_APPLICATION_ERROR (-20001, 'El aula ' || p_aula ||
            'tiene ' || v_numero_ordenadores || ' ordenadores');
    END IF;
    ...
END;
```



## 7.1. Propagación de Excepciones

---

- ❑ Tras generarse una excepción, el sistema busca la rutina de tratamiento de la excepción en la zona de excepciones del bloque PL/SQL que estaba ejecutando. Si no la encuentra, la buscará en el bloque que contiene al bloque en ejecución. Esta búsqueda seguirá hasta encontrar la rutina apropiada o volver al entorno de ejecución
- ❑ Después de encontrar la rutina, se ejecuta ésta y se sale del bloque en el que se ha encontrado para seguir ejecutando normalmente el bloque que lo contiene
- ❑ El gestor OTHERS capturaré cualquier excepción generada. Como la búsqueda del gestor de excepciones se realiza de forma secuencial, el gestor OTHERS debe ser siempre el último gestor de un bloque
- ❑ Resulta muy útil definir un gestor OTHERS en el bloque de nivel superior del programa para asegurarse que no queda ningún error por capturar
- ❑ Para saber que error provocó la excepción dentro del gestor OTHERS podemos usar las funciones SQLCODE y SQLERRM para obtener el código del error y el mensaje asociado

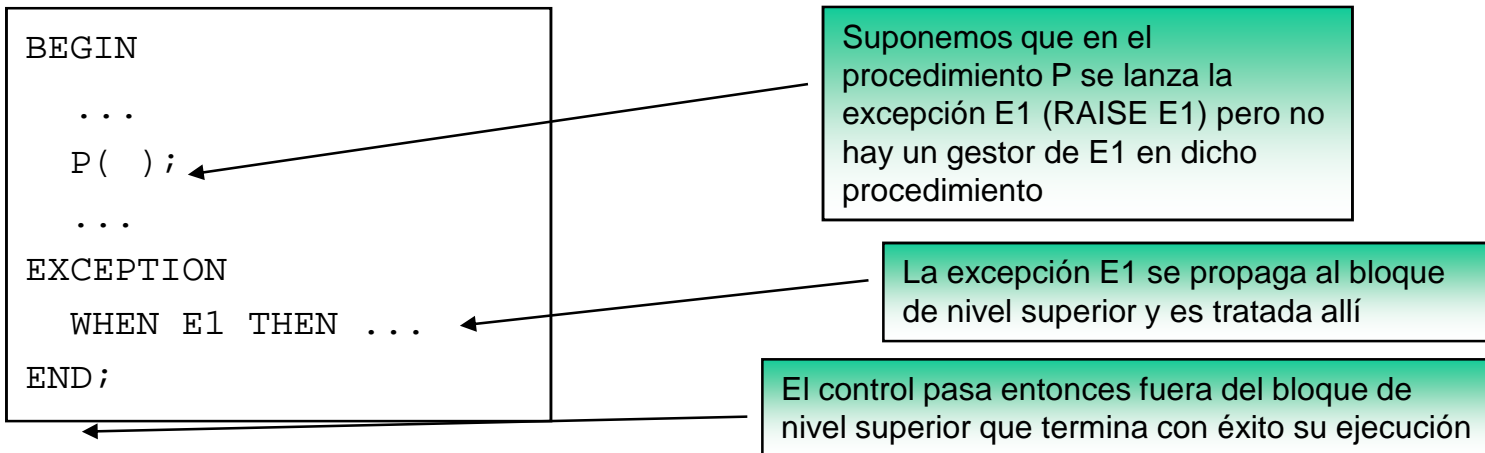
```
WHEN OTHERS THEN
    cod_error := SQLCODE;
    texto_error := SQLERRM;
    INSERT INTO TABLA_ERRORES
        VALUES(cod_error, texto_error, SYSDATE);
```

## 7.1. Propagación de Excepciones

- ❑ Una llamada a un procedimiento también da lugar a un bloque de nivel superior.

```
BEGIN
  -- Inicio del bloque externo
  -- Llamada a un procedimiento. Este bloque externo será el
  -- bloque de nivel superior del procedimiento
  P(...);
END;
```

- ❑ Si el procedimiento P( ) genera una excepción que queda sin tratar, ésta se propaga al bloque externo, que es el de nivel superior con respecto al procedimiento



❑ Ejemplos de excepciones comunes [definidas por el sistema](#):

- **TOO\_MANY\_ROWS**  
una sentencia SELECT.. INTO devolvió más de una tupla
- **NO\_DATA\_FOUND**  
una sentencia SELECT.. INTO no devolvió ninguna tupla
- **ZERO\_DIVIDE**  
intento de división de un número por cero
- **CURSOR\_ALREADY\_OPEN**  
intento de abrir un cursor ya abierto
- **DUP\_VAL\_ON\_INDEX**  
violación de una restricción de unicidad

Estas Excepciones se disparan automáticamente cuando ocurre el evento en cuestión:

EXCEPTION

...

WHEN NO\_DATA\_FOUND THEN

[sentencias](#)

...

- ❑ Los subprogramas (procedimientos y funciones) son bloques con nombre que pueden tener parámetros y ser invocados desde otros bloques PL/SQL

Sintaxis

```
[ CREATE [OR REPLACE] ]  
  
{ PROCEDURE nombre_procedimiento [(parametros)] |  
  
  FUNCTION nombre_funcion [(parametros)] RETURN tipo_dato } { IS | AS }  
  [ declaraciones locales ]  
  
BEGIN  
  sentencias  
[ EXCEPTION  
  tratamiento de excepciones ]  
END [nombre_procedimiento];
```

- ❑ Los parámetros son una lista separada por comas con el formato:

*nombre\_parametro* [ IN | OUT | IN OUT ] *tipo* [{:= | DEFAULT} *expresion*]

- ❑ IN: el valor del parámetro se pasa al hacer la llamada al subprograma, y no es modificado dentro del mismo. Si no se especifica nada es el calificador **por defecto**.
- ❑ OUT: dentro del subprograma el parámetro formal actúa como una variable sin inicializar. Al salir del subprograma, se devuelve un valor en el parámetro especificado.
- ❑ IN OUT: combinación de los modos IN y OUT.
- ❑ En caso de **parámetros** de tipo CHAR, VARCHAR2 o NUMBER no debe especificarse su longitud y/o precisión
- ❑ La sentencia RETURN acaba la ejecución del subprograma inmediatamente devolviendo un resultado si procede (funciones)

## 8. Procedimientos y Funciones

```
Procedure Ejemplo (P1 IN Number, P2 OUT Number, P3 IN OUT Number) IS
  varLocal Number;
BEGIN
  ...
  varLocal:=P1; /*Permitido*/
  P1:=7;        /* No permitido */
  varLocal:=P2; /* varLocal toma valor NULL */
  P2:=5;        /*Permitido*/
  varLocal:= P3; /* Permitido*/
  P3:=8;        /*Permitido*/
  ...
END;
```

```
Declare
v1 number :=1; v2 number :=2; v3 number :=3;

BEGIN
  Ejemplo (v1,v2,v3);
END;
```

Valores tras la llamada a " Ejemplo(v1,v2,v3)"

v1:=1      v2:= 5      v3:=8

### Ejemplo

```
FUNCTION NombreAlumno (idAlumno ALUMNO.nAl%TYPE)
                                RETURN ALUMNO.nombre%TYPE IS
    nomb ALUMNO.nombre%TYPE;
    alumno_muy_antiguo EXCEPTION;
BEGIN
    IF TO_NUMBER(idAlumno) < 100 THEN
        RAISE alumno_muy_antiguo;
    END IF;
    SELECT nombre INTO nomb FROM ALUMNO WHERE nAl = idAlumno;
    RETURN nomb;
EXCEPTION
    WHEN alumno_muy_antiguo THEN
        sentencias para la gestión de alumno muy antiguo;
END NombreAlumno;
```

- ❑ Los subprogramas pueden agruparse en *paquetes* (*packages*), proporcionando ventajas tales como modularidad, facilidad de diseño, encapsulamiento, y mejoras en el rendimiento
- ❑ Oracle proporciona varios paquetes predefinidos de utilidades, que permiten realizar operaciones como:
  - leer y escribir en ficheros
  - proveer acceso a algunas sentencias LDD de SQL
  - ejecutar y controlar procesos en el sistema operativo desde la base de datos
  - mostrar información por consola
  - ✓ el paquete que contiene estas operaciones es el **DBMS\_OUTPUT**



### Ejemplo

Desde SQL Developer, para usar DBMS\_OUTPUT con objeto de mostrar mensajes de depuración, se debería activar primero la salida en pantalla mediante:

```
SQL> SET SERVEROUTPUT ON
```

Y luego insertar en el código PL/SQL llamadas a *put\_line* para mostrar la información deseada:

```
...  
dbms_output.put_line('Valor de contador: ' || TO_CHAR (contador));  
...
```

## 9. Ejemplo Resumen

Mediante un procedimiento almacenado, se quiere mostrar una estadística por cada asignatura, del porcentaje de alumnos aprobados respecto a los presentados en la convocatoria feb\_jun, para un curso y un año académico dados por parámetro. (ejemplo: porcentaje\_aprobados (2, 2002) → 2º curso / año 2002)

```
set serveroutput on;
create or replace
procedure porcentaje_aprobados(p_curso ASIGNATURA.curso%type,
                               p_año MATRICULA.año%type) is

    existe_año integer;
    v_idAsig ASIGNATURA.idAsig%type;
    porcentaje number(5,2);
    num_alumnos_presentados integer;
    num_alumnos_aprobados integer;
    no_existe_año exception;

    C1 { cursor c_asignaturas is
        select idAsig, nombre
        from ASIGNATURA
        where curso = p_curso;

        C2 { cursor c_matriculas(p_idAsig MATRICULA.idAsig%type) is
        select idAsig, feb_jun
        from MATRICULA
        where año = p_año and idAsig=p_idAsig;
```

## 9. Ejemplo Resumen

```
begin
  select count(*) into existe_año from MATRICULA where año = p_año;
  if existe_año = 0 then raise no_existe_año; end if;
  for v_asignaturas in c_asignaturas loop
    num_alumnos_presentados:=0; num_alumnos_aprobados:=0;
    for v_matriculas in c_matriculas(v_asignaturas.idAsig) loop
      if v_matriculas.feb_jun is not null then
        num_alumnos_presentados := num_alumnos_presentados+1;
      if v_matriculas.feb_jun >= 5 then
        num_alumnos_aprobados := num_alumnos_aprobados+1;
      if num_alumnos_presentados <> 0 then
        porcentaje := (num_alumnos_aprobados/num_alumnos_presentados)*100;
      else porcentaje := 0;
      dbms_output.put_line(v_asignaturas.nombre || ' ' ||
                           num_alumnos_presentados || ' ' ||
                           num_alumnos_aprobados || ' ' || porcentaje || '%');
    end loop;
  end loop;
```

The diagram illustrates the execution flow of the PL/SQL code. It shows three nested loops: an outer loop for v\_asignaturas (labeled B1), an inner loop for v\_matriculas (labeled B2), and a final if statement (labeled I3). Within the inner loop B2, there are two conditional branches: one for v\_matriculas.feb\_jun is not null (labeled I1) and another for v\_matriculas.feb\_jun >= 5 (labeled I2). The final if statement I3 branches based on whether num\_alumnos\_presentados is not equal to 0.

```
exception
  when no_existe_año then
    dbms_output.put_line('No hay información del curso ' || p_año ||
                        ' en la BD');
end;
```