



PROGRAMACIÓN PL/SQL

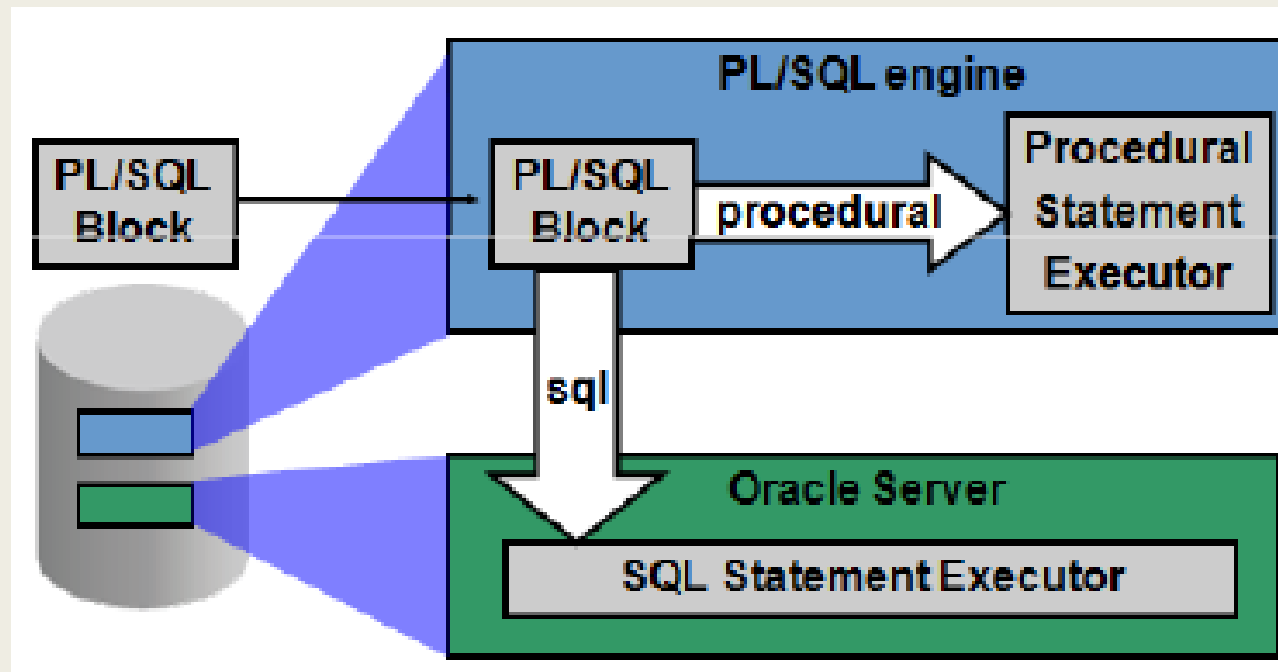


Introducción

- **PL/SQL** (Procedural Language/Structured Query Language).
- Lenguaje de programación incluido en Oracle.
- Sus sentencias son ejecutadas por el SGBD.
- En la actualidad otros SGBD incluyen su propio PL/SQL (con algunas diferencias con respecto al de Oracle)
 - MySQL
 - DB2
 - SQL Server -> TRANSACT SQL

Características

- Los programas creados con PL/SQL se almacenarán en la base de datos como un objeto más, facilitando el acceso a los distintos usuarios.
- Los programas se ejecutan en el lado del servidor.



Características

- Es un lenguaje estructurado y potente, con estructuras ideales para el trabajo con bases de datos.
- Permite minimizar la comunicación entre Cliente y Servidor de BD y los accesos a la BD.
- Permite incluir sentencias SQL (DML).
- Incluye los tipos de datos, operadores y funciones de SQL.



ELEMENTOS



Bloques PL/SQL

- Estructura básica en un programa PL/SQL.

```
DECLARE /*Opcional*/  
    Declaración de objetos, variables, cursores,  
    constantes y excepciones.  
BEGIN /*Obligatorio*/  
    Sentencias SQL.  
    Sentencias PL/SQL.  
EXCEPTION /*Opcional*/  
    Acciones cuando se produce la excepción.  
END; /*Obligatorio*/
```

Bloques PL/SQL

- Ejemplo básico (los bloques también podrían anidarse):

```
DECLARE
    v_num_mecanicos NUMBER;
BEGIN
    SELECT COUNT(*) into v_num_mecanicos
    FROM MECANICOS
    DBMS_OUTPUT.PUT_LINE(v_num_mecanicos);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error');
END;
```

Elementos del lenguaje: Comentarios

- Comienzan por -> --
- Entre -> /* ... */

```
DECLARE
    v_num_mecanicos NUMBER; /*Numero*/
BEGIN
    -- Comentario
    SELECT COUNT(*) into v_num_mecanicos
    FROM MECANICOS;
    DBMS_OUTPUT.PUT_LINE(v_num_mecanicos);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error');
END;
```


Elem. Lenguaje: Palabras reservadas

- Tienen un significado especial:

- *DECLARE*

- *BEGIN*

- *IF*

- *NUMBER*

- *...*

Elem. Lenguaje: Identificadores

- Los identificadores son secuencias de caracteres (nombres) que se utilizan para hacer referencia a componentes del programa (variables, constantes, funciones, tipos de datos, etc.).
- Deben empezar por una letra y pueden tener hasta 30 caracteres.
- No se diferencia entre mayúsculas y minúsculas (como en SQL).

Elem. Lenguaje: Tipos de datos

Tipos de datos compuestos

- Tablas
- Registros
- Objetos

Subtipos

- Se pueden definir nuevos tipos restringiendo los tipos numéricos básicos

```
SUBTYPE nombre_subtipo IS  
tipo_base [RANGE min..max] [NOT NULL];
```

```
DECLARE
```

```
    SUBTYPE digitos IS NUMBER RANGE 1..9;
```

Elem. Lenguaje: Constantes

- Son identificadores que no cambian su valor.
- Deben inicializarlas en el momento de declararlas.

```
nombre CONSTANT Tipo := valor;
```

- Ejemplo:

```
DECLARE
```

```
    salario_base CONSTANT NUMBER := 900;
```

Elem. Lenguaje: Variables

- Las variables deben declararse en el bloque correspondiente y siempre antes de su uso.

```
nombre Tipo [NOT NULL] [:= | DEFAULT valor];
```

Nota: NOT NULL obliga a que la variable tenga siempre un valor. Es obligatorio inicializarla (con := o con DEFAULT).

DECLARE

```
v_salario1 NUMBER := 900;  
v_salario2 NUMBER DEFAULT 900;  
v_nombre VARCHAR2 := 'Raquel';  
fin BOOLEAN := TRUE;  
edad NUMBER NOT NULL := 30;
```

Elem. Lenguaje: Variables

■ %TYPE

- Permite declarar una variable del mismo tipo que otra variable o campo de tabla.
- Permite programas mas robustos frente a cambios.

```
DECLARE
```

```
    v_salario1 mecanicos.salario%TYPE;
```

```
    v_salario2 v_salario1%TYPE;
```

Elem. Lenguaje: Variables

■ %ROWTYPE

- Crea una variable de tipo registro (struct en C) cuyos campos se corresponden con los campos de una tabla o vista.
- Para hacer referencia a cada uno de esos campos se emplea la notación **<Variable>.<Campo>**.

```
DECLARE
    v_coche coches%ROWTYPE;
BEGIN
    ...
    v_coche.matricula;
    ...
```

Elem. Lenguaje: Variables

- Ámbito de una variable (donde es visible)
 - Bloque donde se declara y sus bloques hijos.
 - La variable será local para el bloque en el que se ha declarado y global para los bloques hijos.
 - Las variables declaradas en los bloques hijos no serán visibles desde el bloque padre.

Elem. Lenguaje: Operadores

■ Operadores de SQL

- Asignación: `:=`
- Lógicos: `AND`, `OR`, `NOT`
- Concatenación: `||`
- Comparación: `<`, `>`, `=`, `!=`, `>=`, `<=`, `IS NULL`, `BETWEEN`, `LIKE`, `IN`, ...
- Aritméticos: `+`, `-`, `*`, `/`, `**`, ...
- Fechas: Pueden restarse ($f1-f2$) y también se les puede sumar o restar un número n , de días a la fecha ($f1 \pm n$).

Elem. Lenguaje: Variables


- Ámbito de una variable (donde es visible)
 - Bloque donde se declara y sus bloques hijos.
 - La variable será local para el bloque en el que se ha declarado y global para los bloques hijos.
 - Las variables declaradas en los bloques hijos no serán visibles desde el bloque padre.

Elem. Lenguaje: Variables

- Ámbito de una variable (donde es visible)
 - Bloque donde se declara y sus bloques hijos.
 - La variable será local para el bloque en el que se ha declarado y global para los bloques hijos.
 - Las variables declaradas en los bloques hijos no serán visibles desde el bloque padre.

Elem. Lenguaje: Variables

- Ámbito de una variable (donde es visible)
 - Bloque donde se declara y sus bloques hijos.
 - La variable será local para el bloque en el que se ha declarado y global para los bloques hijos.
 - Las variables declaradas en los bloques hijos no serán visibles desde el bloque padre.

A thick black L-shaped frame is positioned on the left and bottom edges of the slide, framing the central text.

ESTRUCTURAS DE CONTROL

Estruct. Control Condicionales: IF

■ Simple

```
IF <cond> THEN  
    <sentencias1>;  
END IF;
```

■ Doble

```
IF <cond> THEN  
    <sentencias1>;  
ELSE  
    <sentencias2>;  
END IF;
```

Estruct. Control Condicionales: IF

■ Múltiple

```
IF <cond1> THEN
    <sentencias1>;
ELSIF <cond2> THEN
    <sentencias2>;
...

[ELSE
    <sentenciasN>;
]
END IF;
```

Estruct. Control Condicionales: IF

■ Ejemplo

```
DECLARE
    v_num_mecanicos NUMBER;
BEGIN
    SELEC CONUNT(*) into v_num_mecanicos
    FROM MECANICOS;
    IF (v_num_mecanicos =0) THEN
        DBMS_OUTPUT.PUT_LINE('No hay mecanicos');
    ELSE
        DBMS_OUTPUT.PUT_LINE(v_num_mecanicos);
    END IF;
END;
```


Estruct. Control Condicionales: CASE

■ Simple

```
CASE <selector>
  WHEN <valor1> THEN
    <sentencias1>;
  WHEN <valor2> THEN
    <sentencias2>;
  ...
  WHEN <valorn> THEN
    <sentenciasN>;
  [ELSE
    <sentencias_else>;
  ]
END CASE;
```

Estruct. Control Condicionales: CASE

■ Ejemplo

```
DECLARE
    nota CHAR(1);
BEGIN
    nota := 'B';
    CASE nota
    WHEN 'A' THEN
        DBMS_OUTPUT.PUT_LINE('Excelente');
    WHEN 'B' THEN
        DBMS_OUTPUT.PUT_LINE(' Muy bien');
    WHEN 'C' THEN
        DBMS_OUTPUT.PUT_LINE('Bien');
    WHEN 'D' THEN
        DBMS_OUTPUT.PUT_LINE('Mal');
    ELSE
        DBMS_OUTPUT.PUT_LINE('No evaluado');
    END CASE;
END;
```

Estruct. Control Condicionales: WHILE

■ Definición

```
WHILE cond LOOP  
    <sentencias>;  
END LOOP;
```

```
DECLARE  
    i NUMBER;  
BEGIN  
    i:=0;  
    WHILE i<3 LOOP  
        DBMS_OUTPUT.PUT_LINE (i);  
        i:=i+1;  
    END LOOP;  
END;
```

Estruct. Control Condicionales: FOR

- La variable de control (<var>) no hay que declararla y es local al propio bucle no pudiendo utilizarse fuera del mismo.
- El incremento siempre será de 1 (no se permite otro valor).

```
FOR <var> IN [ REVERSE ] <valIn> <valFin> LOOP  
    <sentencias>;  
END LOOP;
```

- Para decrementar se utiliza la opción REVERSE y entonces empezará por <ValorFin> hasta llegar a <ValorIni>.

Estruct. Control Condicionales: FOR

- Ejemplo: Contador ascendente y descendente de 1 a 3, mostrado por pantalla.

```
BEGIN
  FOR i IN 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;
  FOR i IN REVERSE 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;
END;
```

Estruct. Control Condicionales: FOR

- También se puede utilizar para recorrer los registros resultantes de una consulta incluyendo la consulta en la clausula IN.

```
FOR <var> IN [ REVERSE ] (consulta) LOOP  
    <sentencias>;  
END LOOP;
```

```
BEGIN  
    FOR i IN (SELECT SUELDO FROM MECANICOS) LOOP  
        DBMS_OUTPUT.PUT_LINE (i.sueldo);  
    END LOOP;  
END;
```

Estruct. Control Condicionales: LOOP

- LOOP ofrece bucles de los que se sale mediante "EXIT".

```
LOOP
    <sentencias1>;
    IF cond THEN EXIT;
    END IF;
    <sentencias2>;
END LOOP;
```

```
DECLARE
    x NUMBER := 0;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE (TO_CHAR(x));
        x := x + 1;
        IF x > 3 THEN
            EXIT;
        END IF;
    END LOOP;
END;
```



ELEMENTOS DEL LENGUAJE



PL/SQL: Sentencias DML

- SELECT ... INTO
 - Detrás del INTO
 - Lista de tantas variables como elementos seleccionados y con sus tipos compatibles en el mismo orden,
 - O, un registro con sus campos también compatibles
 - La consulta solo puede recuperar una fila.
 - Para consultas que retornan mas de una fila hay que usar cursores
- INSERT/UPDATE/DELETEDetrás del INTO

PL/SQL: Sentencias DML

■ Ejemplo:

```
DECLARE
    v_marca coches.marca%TYPE;
BEGIN
    SELECT marca into v_marca
    FROM coches
    WHERE matricula='M3020KY';
    DBMS_OUTPUT.PUT_LINE(v_marca);
END;
```

PL/SQL: Sentencias DML

```
DECLARE
    v_marca coches.marca%TYPE;
    v_modelo coches.modelo%TYPE;
BEGIN
    SELECT marca,modelo into v_marca,v_modelo
    FROM coches
    WHERE matricula='M3020KY';

    INSERT INTO COCHES
    VALUES ('M1111UU',v_marca, v_modelo,2008);
END;
```

PL/SQL: Sentencias DML

```
DECLARE
    v_incremento mecanicos.salario%TYPE:=100;
BEGIN
    UPDATE mecanicos
    SET salario = salario + v_incremento
    WHERE puesto = 'CHAPA';
END;
```

PL/SQL: Sentencias DML

```
DECLARE
    v_modelo coches.modelo%TYPE;
BEGIN
    SELECT modelo into v_modelo
    FROM coches
    WHERE matricula='M3020KY';

    DELETE FROM COCHES
    WHERE modelo=v_modelo;
END;
```

The image features two thick, black L-shaped brackets. One is positioned on the left side, with its horizontal bar at the top and its vertical bar extending downwards. The other is on the right side, with its vertical bar at the top and its horizontal bar at the bottom. These brackets frame the central text.

INTERACCIÓN CON EL USUARIO

Interacción con el usuario

- PL/SQL no está preparado para interactuar con el usuario.
- Está pensado para interactuar con la BBDD.
- Para depurar y probar programas existe el paquete DBMS_OUTPUT (se explicará más adelante).
 - Ofrece funciones sencillas que permiten mostrar resultados por pantalla
 - Sintaxis: **DBMS_OUTPUT.PUT_LINE**(expresión);

Interacción con el usuario

- Para que funcione correctamente hay que habilitar en el cliente desde donde se lanza el programa PL/SQL la variable de entorno SERVEROUTPUT:
 - SET SERVEROUTPUT ON;

```
SET SERVEROUTPUT ON;  
DECLARE  
    marca1 coches.marca%TYPE;  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('Hola');  
END;
```


Interacción con el usuario

- Variables de sustitución
 - Se pueden usar en bloques anónimos
 - El programa pide valores para las variables indicadas “de sustitución” (&variable)

```
DECLARE
    v_salario mecanicos.salario%TYPE;
BEGIN
    SELECT salario INTO v_salario
    FROM mecanicos
    WHERE dni=&v_dni;
    DBMS_OUTPUT.PUT_LINE(v_salario);
END;
```



PROGRAMACIÓN PL/SQL (II)



Introducción

- La estructura básica de un programa PL/SQL es el **bloque**.
- La utilización de los bloques permite una mejora de rendimiento ya que reduce el número de operaciones de E/S al enviar al servidor el bloque completo y no instrucciones sueltas.
- Los bloques pueden anidarse.
- Normalmente cada bloque realiza una unidad lógica de trabajo.

Tipos de bloque

- **Bloques anónimos.**
- **Bloques con nombre** (iguales que los anónimos pero con una etiqueta que les asigna un nombre)
- **Subprogramas**
 - **Funciones.**
 - **Procedimientos.**
- **Triggers (disparadores).**





BLOQUES ANÓNIMOS

Bloques anónimos

- No tienen nombre.
- No se almacenan en la BD.
- Se ejecutan tras escribirlos.

```
DECLARE
    v_num_mecanicos NUMBER;
BEGIN
    SELECT COUNT(*) into v_num_mecanicos
    FROM MECANICOS
    DBMS_OUTPUT.PUT_LINE(v_num_mecanicos);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error');
END;
```



BLOQUES CON NOMBRE

Bloques anónimos

- Se puede poner una etiqueta a un bloque anónimo para facilitar la referencia a él.
- En **bloques anidados puede ser útil nombrarlos para referirse a una** variable en particular de ese bloque, en caso de que existan varias con el mismo nombre

```
<<BloqueContarMecanicos>>
DECLARE
    v_num_mecanicos NUMBER;
BEGIN
    SELECT COUNT(*) into v_num_mecanicos
    FROM MECANICOS
    DBMS_OUTPUT.PUT_LINE(v_num_mecanicos);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error');
END BloqueContarMecanicos;
```

– *BloqueContarMecanicos.v_num_mecanicos*



PROCEDIMIENTOS Y FUNCIONES



Introducción

- Se definen como subprogramas
 - Almacenados
 - Compilados en la base de datos.
- Tienen nombre.
- Pueden ser invocados/llamados desde otros bloques PL/SQL.



PROCEDIMIENTOS



Procedimientos

- Pueden recibir y devolver múltiples parámetros

```
CREATE [OR REPLACE] PROCEDURE Nombre[(<lista de parámetros>)] IS|AS
    Declaración de objetos, variables, cursores, constantes y
    excepciones.
BEGIN
    Sentencias SQL.
    Sentencias PL/SQL.
EXCEPTION /*Opcional*/
    Acciones cuando se produce la excepción.
END [Nombre];
```

Procedimientos

- Pueden llamar a otros procedimientos o funciones.

```
CREATE OR REPLACE PROCEDURE AumentarSalarioPorPuesto(v_aumento
IN NUMBER, v_puesto IN VARCHAR) IS
BEGIN
    UPDATE mecanicos
    SET salario:=salario + v_aumento
    WHERE puesto = v_puesto;
END;
```

- Borrar un procedimiento

```
DROP PROCEDURE Nombre;
```



FUNCIONES

Funciones

- Pueden recibir parámetros y retornar un valor “en su nombre”.

```
CREATE [OR REPLACE] FUNCTION Nombre[(<lista de parámetros>)]  
RETURN Tipo IS|AS  
    Declaración de objetos, variables,  
    cursores, constantes y excepciones  
BEGIN  
    [Sentencias SQL/PLSQL]  
    RETURN <expresión>;  
    [Sentencias SQL/PLSQL]  
EXCEPTION /*Opcional*/  
    Acciones cuando se produce la excepción.  
END [Nombre];
```

Funciones

- El **RETURN** de la cabecera especifica el tipo de valor devuelto por la función.
- El/los **RETURN** del cuerpo de la función devuelven el valor indicado a continuación y pasa el control de ejecución al programa llamante (programa principal).
- Pueden llamar a otros procedimientos o funciones.
- Las funciones se invocan desde: Un bloque anónimo, un procedimiento u otra función.
- Para borrar una función:

```
DROP FUNCTION Nombre;
```


Funciones

■ Ejemplo de definición de función

```
CREATE OR REPLACE FUNCTION NumeroMecanicosPorPuesto(v_puesto
IN VARCHAR) RETURN NUMBER
IS
    v_numero NUMBER := 0;
BEGIN
    SELECT count(*) into v_numero
    FROM mecanicos
    WHERE puesto=v_puesto;
    RETURN v_numero;
END;
```

Funciones

- Ejemplo de bloque anónimo llamante

```
DECLARE
    numero NUMBER;
BEGIN
    numero:=NumeroMecanicosPorPuesto('CHAPA');
    DBMS_OUTPUT.PUT_LINE(numero);
END;
```

Funciones: Condiciones especiales

- Para que las funciones puedan ser invocadas desde **SQL**, éstas tienen que cumplir que:
 - Sólo pueden utilizar parámetros de tipo IN.
 - Sus parámetros deben ser de tipos compatibles con el lenguaje SQL (no valen tipos específicos de PL/SQL como BOOLEAN por ejemplo).
 - No pueden utilizar instrucciones de transacciones (COMMIT, ROLLBACK,...)
 - Si una instrucción DML modifica una determinada tabla, en dicha instrucción no se puede invocar a una función que realice consultas sobre la misma tabla.



PARÁMETROS

Parámetros: Formato

- <nombre>[IN|OUT|IN OUT]<tipo>[{:=|DEFAULT}<valor>]

```
CREATE OR REPLACE PROCEDURE NumeroMecanicosPorPuesto(v_puesto
IN VARCHAR, v_numero OUT NUMBER) IS
BEGIN
    v_numero NUMBER := 0;
    SELECT count(*) into v_numero
    FROM mecanicos
    WHERE puesto=v_puesto;
END;
```

Nota: Si se omite el tipo (IN, OUT, IN OUT) se considera IN por defecto.

Parámetros: Tipos

■ IN

- Parámetro de entrada.
- Paso por valor.
- Sólo puede aparecer en el lado derecho de una asignación.
- El parámetro puede ser una constante.

■ OUT

- Parámetro de salida
- Sólo puede aparecer en el lado izquierdo de una asignación (para tomar un valor).
- El parámetro actual debe ser una variable.

Parámetros: Tipos (II)

■ IN OUT

- Parámetro de entrada salida.
- Paso de parámetros por referencia.
- Permite pasar un valor inicial y devolver un valor actualizado.
- Puede aparecer en ambos lados en una asignación.
- El parámetro actual (cuando se llama al subprograma) debe ser una variable.

En la mayoría de los lenguajes de programación hay dos formas de pasar las variables a una función, **por valor** o **por referencia**. Por valor significa que la función (o subrutina) recibe sólo una copia del valor que tiene la variable, o sea que no la puede modificar. Por referencia recibe la propia variable pasada desde el programa llamante.

Parámetros: Entrada de datos

- Cuando se definen los parámetros no pueden tener restricciones de tamaño:
 - NO -> VARCHAR2(10)
 - SI -> VARCHAR2.
- Pueden producirse errores si en la definición del subprograma no se tienen en cuenta las posibles restricciones de los argumentos que se utilicen en la llamada. Ejemplo:
 - Se produce un error (ORA-6502) si se asigna una cadena de longitud 5 a un argumento formal tipo OUT que en la llamada tenga una variable de tipo VARCHAR2(2).

Parámetros: Notación

- Existen dos formas de pasar argumentos a un procedimiento/función cuando se invoca.
 - **Notación posicional:** Se pasan los valores de los parámetros en el mismo orden en que se han definido.

```
BEGIN  
    SubirSuelo(101,100);  
END;
```

- **Notación nominal:** Se pasan los valores en cualquier orden nombrando explícitamente el parámetro.

```
BEGIN  
    SubirSuelo(suelo => 100,dni => 101);  
END;
```

Compilación y estado

- Cuando se crea (CREATE) un procedimiento/función Oracle lo compila automáticamente y lo almacena.
- Los procedimiento/ función pueden encontrarse en dos estados:
 - **Valid**: cuando el procedimiento/función y los procedimiento/función referenciados no se han modificado desde la última compilación.
 - **Invalid**: cuando se han modificado (o borrado) procedimientos/funciones referenciados desde la última compilación.

Compilación y estado (II)

- **Ver el estado y código**
 - Vista **USER_OBJECTS**
 - **Ejemplo**

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
FROM USER_OBJECTS
WHERE OBJECT_TYPE='PROCEDURE' OR
      OBJECT_TYPE='FUNCTION';
```

Compilación y estado (II)

- **Código fuente original de cada subprograma**

- Vista **USER_SOURCE**

```
SELECT text  
FROM USER_SOURCE  
WHERE name= 'NumeroMecanicosPorPuesto';
```

- **Información sobre los errores de compilación.**

- Vista **USER_ERRORS:**

- **Compilar explícitamente un procedimiento / función**

- ALTER {PROCEDURE | FUNCTION} Nombre COMPILE;

Privilegios

- Para que un usuario pueda ejecutar un subprograma ha de **concedérsele permiso EXECUTE sobre ese objeto**:

```
GRANT EXECUTE ON <NombreSubprograma>  
TO <Usuario>;
```

- Para compilarse, un subprograma necesita permisos sobre todos los objetos que utilice.

Privilegios

- Si no son suyos debe tener concedido el permiso de forma explícita, no a través de un rol, ya que un usuario puede desactivar un rol (SET ROLE) en una sesión.
- Un subprograma se ejecuta como si lo ejecutara su propietario, por lo que por ejemplo, el subprograma no usará las tablas del usuario que lo ejecute aunque se llamen igual que las tablas del propietario del subprograma.

Funciones/procedimientos locales

- Los procedimientos/funciones locales se declaran al final de la zona de declaraciones de otro bloque.
- Siguen las mismas reglas de ámbito que para las variables.
- Se emplean cuando el subprograma no va a ser reutilizado.
- Si se hacen referencias cruzadas entre funciones (con llamadas mutuamente recursivas) deben hacerse declaraciones anticipadas.
- Dentro de un procedimiento/función se puede declarar otro.

Funciones/procedimientos locales (II)

```
CREATE [OR REPLACE] PROCEDURE Nombre[(<lista de parámetros>)] IS|AS
    Declaración de objetos, variables, cursores, constantes y
    excepciones.
    PROCEDURE NombreLocal[(<lista de parámetros>)] IS|AS
        Declaraciones
    BEGIN
        Sentencias
    EXCEPTION /*Opcional*/
        Acciones cuando se produce la excepción.
    END [NombreLocal];
BEGIN
    [Sentencias]
    NombreLocal (...);
    [Sentencias]
EXCEPTION /*Opcional*/
    Acciones cuando se produce la excepción.
END [Nombre];
```


Recursividad

- PL/SQL permite recursividad.

```
CREATE OR REPLACE FUNCTION Factorial(n NUMBER) RETURN NUMBER IS
BEGIN
    IF n=1 THEN
        RETURN 1;
    ELSE
        RETURN n*Factorial(n-1);
    END IF;
END Factorial;
```

Consulta de información

■ Vistas del diccionario de datos:

- **DBA_PROCEDURES.**
- **ALL_PROCEDURES.**
- **USER_PROCEDURES.**
- **DBA_FUNTIONS.**
- **ALL_FUNTIONS.**
- **USER_FUNTIONS.**



PROGRAMACIÓN PL/SQL (III)





CURSORES

Introducción

- Los cursores **son áreas de memoria** donde se **almacena información extraída de la base de datos**.

Tipos

■ **Cursores implícitos.**

- Declarados y creados implícitamente en SELECT ... INTO que devuelven una sola fila.
- Declarados y creados implícitamente en todas las sentencias INSERT, DELETE y UPDATE .

■ **Cursores explícitos.**

- Declarados y nombrados por el programador.
- Se manipulan con sentencias específicas.
- Utilizados en SELECT que devuelven más de 1 fila.



IMPLÍCITOS

Cursores implícitos: Introducción

- Declarados implícitamente y creados cuando se realizan operaciones DML.
- Se crean y destruyen de forma transparente para el usuario.

Cursores implícitos: Excepciones

Declarados implícitamente en SELECT que devuelven una sola fila.

SELECT ... INTO ...

- Si la consulta no devuelve nada se produce una excepción.
 - **NO_DATA_FOUND**
- Si la consulta devuelve mas de una fila se produce una excepción.
 - **TOO_MANY_ROWS.**

Cursores implícitos: Excepciones

```
SET SERVEROUTPUT ON;
DECLARE
    v_marca coches.marca%type;
BEGIN
    SELECT marca into v_marca
    FROM coches
    WHERE matricula='M3020KY';
    DBMS_OUTPUT.PUT_LINE('Marca ' || v_marca);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No hay coches con esa matricula');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Hay mas de un coche con esa matrícula');
END;
```

Cursores implícitos: Excepciones

```
SET SERVEROUTPUT ON;
DECLARE
    v_coche coches%rowtype;
BEGIN
    SELECT * into v_coche
    FROM coches
    WHERE matricula='M3020KY';
    DBMS_OUTPUT.PUT_LINE('Marca ' || v_coche.marca);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No hay coches con esa matricula');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Hay mas de un coche con esa matrícula');
END;
```

Cursores implícitos: Delete y Update

- Declarados implícitamente en todas las sentencias DELETE y UPDATE .

```
CREATE OR REPLACE PROCEDURE AumentarSalarioPorPuesto(v_aumento
IN NUMBER, v_puesto IN VARCHAR) IS
BEGIN
    UPDATE mecanicos
    SET salario:=salario + v_aumento
    WHERE puesto = v_puesto;
END;
```

Cursores implícitos: Atributos

Existen una serie de atributos sobre los cursores implícitos.

■ **SQL%ISOPEN**

- Devuelve FALSE siempre ya que Oracle cierra automáticamente el cursor implícito después de cada orden SQL.

■ **SQL%ROWCOUNT**

- Devuelve el número de filas afectadas por la última operación SELECT INTO, UPDATE, INSERT o DELETE.

Cursores implícitos: Atributos

■ **SQL%NOTFOUND**

- Devuelve TRUE si el último SELECT INTO, UPDATE, INSERT o DELETE han fallado (no han afectado a ninguna fila).

■ **SQL%FOUND**

- Devuelve TRUE si el último SELECT INTO, UPDATE, INSERT o DELETE ha afectado a una o más filas.

Cursores implícitos: Atributos

```
SET SERVEROUTPUT ON;
DECLARE
BEGIN
    UPDATE mecanicos SET salario=salario + 10
    WHERE dni = '1008' ;
    IF SQL%NOTFOUND THEN
        INSERT INTO MECANICOS VALUES ('1008','LOPEZ
        LOPEZ','JUAN','MOTOR',700,'1', '976666) ;
    END IF;
END;
```

Cursores implícitos: Particularidades

- Las excepciones `NO_DATA_FOUND` y `TOO_MANY_ROWS` sólo se activan (`RAISE`) cuando se acaba de ejecutar un “`SELECT INTO`”, pero no se activan cuando se trata de un “`INSERT`, `UPDATE` o `DELETE`”.
- Después de un `SELECT INTO` no tiene sentido preguntar por `SQL%NOTFOUND` ya que si la consulta no devolviera datos se activaría automáticamente la excepción `NO_DATA_FOUND`.
- Cuando un `SELECT INTO` hace referencia a una función de grupo nunca se activará una excepción `NO_DATA_FOUND` y `SQL%FOUND` siempre será verdad ya que las funciones de grupo siempre retornan algún valor (aunque sea `NULL`).



EXPLÍCITOS

Cursores explícitos: Introducción

- Utilizados en SELECT que devuelven más de 1 fila.
- Declarados y nombrados por el programador.
- Se manipulan con sentencias específicas.

Cursores explícitos: Operaciones

■ Declaración

- En la zona de declaraciones del bloque (DECLARE).

```
CURSOR <Nombre> IS  
SELECT <Select>;
```

Cursores explícitos: Operaciones

■ Apertura

- En la zona de instrucciones (BEGIN). `OPEN <Nombre>;`
- Implica la ejecución del SELECT asociado, guardando el resultado en el cursor.
- El cursor **se queda apuntando a la primera fila.**
- **Si la consulta no devuelve ninguna fila no se producirá ninguna EXCEPTION.**
- Hay que utilizar los atributos del cursor para comprobar los resultados obtenidos tras una recuperación.
- Hay un máximo número de cursores que pueden estar abiertos a la vez en la BD.

Cursores explícitos: Operaciones

■ **FETCH (lectura de la información)**

- Cada fetch recupera una fila del SELECT asociado al cursor y avanza automáticamente hasta la fila siguiente.

```
FETCH <Nombre> INTO {<variable>[, <variable>]};
```

- En la variable o lista de variables se guarda la información correspondiente a la fila leída del cursor.
- Si se guarda toda la fila en una variable, esta se puede declarar en la zona de declaraciones con un ROWTYPE.

```
<var> <Nombre>%ROWTYPE;
```

Cursores explícitos: Operaciones

■ **FETCH (lectura de la información)**

- El resultado de un FETCH debe comprobarse siempre para controlar cuando llegamos al final de los datos contenidos en el cursor (última fila generada por la consulta).
- Si se realiza un FETCH cuando ya no quedan filas por leer no se producirá ningún error y las variables conservarán el valor previo que tuvieran.

Cursores explícitos: Operaciones

- **CLOSE (cierre del cursor)**
- Cuando ya no se va a utilizar. `CLOSE <Nombre>;`
- Desactiva el cursor y libera la memoria reservada.
- Hay que volver a abrir el cursor si quiere utilizar de nuevo.
- No se pueden recuperar datos del cursor una vez cerrado.
- Si se intenta cerrar un cursor que ya está cerrado se produce el error ORA-1001.

Cursores explícitos: Operaciones

```
SET SERVEROUTPUT ON;
DECLARE
    v_coche coches%rowtype;
    cursor c coches is
        select * from coches where marca='PEUGEOT';
BEGIN
    OPEN c;
    FETCH c INTO v_coche;
    WHILE c%FOUND
    LOOP
        DBMS_OUTPUT.PUT_LINE('Matricula ' || v_coche.matricula);
        DBMS_OUTPUT.PUT_LINE('Marca ' || v_coche.marca);
        DBMS_OUTPUT.PUT_LINE('Modelo ' || v_coche.modelo);
        FETCH c INTO v_coche;
    END LOOP;
    CLOSE c;
END;
```


Cursores explícitos: Atributos

Se dispone de cuatro atributos que permiten conocer el estado actual del cursor. `Nombre%Atributo`

■ **%ISOPEN:**

- Devuelve TRUE si el cursor está abierto.
- En caso contrario devuelve FALSE.

■ **%FOUND:**

- Devuelve TRUE si el último FETCH ha recuperado algún valor. FALSE en otro caso.
- Si el cursor no está abierto devuelve NULL.
- Se emplea como condición para saber si aún quedan datos que leer en el cursor.

Cursores explícitos: Atributos

■ **%NOTFOUND**

- Contrario al anterior.
- Devuelve TRUE cuando el último fetch no ha devuelto información.

■ **%ROWCOUNT:**

- Devuelve el número de filas devueltas hasta el momento por el cursor.
- Genera un ERROR si el cursor no está abierto.

Cursores explícitos: Atributos

```
SET SERVEROUTPUT ON;
DECLARE
    v_coche coches%rowtype;
    cursor c coches is
        select * from coches where marca='PEUGEOT';
BEGIN
    IF NOT c coches%ISOPEN THEN
        OPEN c coches;
    END IF;
    FETCH c coches INTO v_coche;
    WHILE c coches%FOUND
    LOOP
        DBMS_OUTPUT.PUT_LINE('Matricula ' || v_coche.matricula);
        DBMS_OUTPUT.PUT_LINE('Marca ' || v_coche.marca);
        DBMS_OUTPUT.PUT_LINE('Modelo ' || v_coche.modelo);
        FETCH c coches INTO v_coche;
    END LOOP;
    CLOSE c coches;
END;
```

Cursores explícitos: Uso de variables

```
create or replace PROCEDURE ListadoCochesPorMarca(v_marca IN VARCHAR)
IS
    v_coche coches%rowtype;
    cursor c_coches is
        select * from coches where marca=v_marca;
BEGIN
    IF NOT c_coches%ISOPEN THEN
        OPEN c_coches;
    END IF;
    FETCH c_coches INTO v_coche;
    WHILE c_coches%FOUND
    LOOP
        DBMS_OUTPUT.PUT_LINE('Matricula ' || v_coche.matricula);
        DBMS_OUTPUT.PUT_LINE('Marca ' || v_coche.marca);
        FETCH c_coches INTO v_coche;
    END LOOP;
    CLOSE c_coches;
END;
```

Cursores explícitos: Simplificar FOR...LOOP

- **FOR ... LOOP** permite simplificar la gestión de los cursores explícitos al automatizar las tareas necesarias (**apertura, fetch, cierre**), con **excepción de la declaración del cursor**. El formato es:

```
<declaración del cursor>  
FOR v_registro IN Nombre LOOP  
    <sentencias>  
END LOOP;
```

- La variable **v_registro** se declara implícitamente (será de tipo NombreCursor%ROWTYPE)
- El bucle terminará automáticamente cuando se haya procesado el último registro contenido en el cursor.

Cursores explícitos: FOR...LOOP

```
CREATE OR REPLACE PROCEDURE ListadoCochesPorMarca(v_marca IN
VARCHAR) IS
    cursor c_coches is
        select * from coches where marca=v_marca;
BEGIN
    FOR v_coche IN c_coches LOOP
        DBMS_OUTPUT.PUT_LINE('Matricula ' || v_coche.matricula);
        DBMS_OUTPUT.PUT_LINE('Marca ' || v_coche.marca);
        DBMS_OUTPUT.PUT_LINE('Modelo ' || v_coche.modelo);
    END LOOP;
END;
```

Cursores explícitos: Con parámetros

Es posible definir parámetros en los cursores.

■ Declaración

```
CURSOR <Nombre>[(lista_paramétros] IS  
SELECT <Select con los parámetros>;
```

- Formato de los parámetros

```
Nombre [IN] <Tipo [{:= | DEFAULT} valor]
```

- Los parámetros serán siempre de entrada (IN).
- Su ámbito se limita al interior del cursor.

Cursores explícitos: Con parámetros

■ Apertura

```
Open Nombre(p1,p2, ...);  
FOR v_registro IN Nombre(p1,p, ...) LOOP  
    <sentencias>  
END LOOP;
```


Cursores explícitos: Con parámetros

```
CREATE OR REPLACE PROCEDURE ListadoCochesPorMarca IS
  cursor c_coches(v_marca IN VARCHAR) is
    select * from coches where marca=v_marca;
BEGIN
  FOR v_coche IN c_coches('TOYOTA') LOOP
    DBMS_OUTPUT.PUT_LINE('Matricula ' || v_coche.matricula);
    DBMS_OUTPUT.PUT_LINE('Marca ' || v_coche.marca);
    DBMS_OUTPUT.PUT_LINE('Modelo ' || v_coche.modelo);
  END LOOP;

  FOR v_coche IN c_coches('PEUGEOT') LOOP
    DBMS_OUTPUT.PUT_LINE('Matricula ' || v_coche.matricula);
    DBMS_OUTPUT.PUT_LINE('Marca ' || v_coche.marca);
    DBMS_OUTPUT.PUT_LINE('Modelo ' || v_coche.modelo);
  END LOOP;
END;
```



PROGRAMACIÓN PL/SQL (IV)



Introducción

- Durante la **ejecución de programas en PL/SQL** pueden ocurrir errores:
 - Se divide por cero.
 - Se intenta insertar en una tabla una fila duplicada.
 - Se produce un error de memoria.
 - ...
- Es necesario poder notificar cuándo se ha producido un error en la ejecución de un programa y salir airoosamente de la situación.

Introducción

- PL/SQL ofrece mecanismos basados en gestión de **excepciones** para **controlar y gestionar** adecuadamente los **errores en tiempo de ejecución**.
- Oracle asigna códigos a los posibles errores en tiempo de ejecución
 - Ej: ORA-2312



EXCEPCIONES

Definición

- Una **excepción** es un mal funcionamiento (o situación anómala) que sucede durante la ejecución de un programa que como consecuencia de que se produzca:
 - Un **error en tiempo de ejecución** detectado por Oracle.
 - Qué **el propio programador lance/active (RAISE) la excepción** (la provoca explícitamente).
- Hay que gestionar adecuadamente las excepciones para que los programas funcionen adecuadamente.

Tratamiento de una excepción

- Cuando se prevé que un código puede dar lugar una posible excepción que se quiere tratar, se **captura la excepción** y se realiza el tratamiento adecuado.

```
...  
EXCEPTION  
    WHEN <NombreEx1>[OR <NombreEx2> OR ...] THEN  
        Sentencias;  
    [WHEN <NombreExA>[OR <NombreExB> OR ...] THEN  
        Sentencias;  
    ...  
    [WHEN OTHERS THEN]  
        Sentencias;  
END;
```

Tratamiento de una excepción

```
SET SERVEROUTPUT ON;
DECLARE
    v_coche coches%rowtype;
BEGIN
    SELECT * into v_coche
    FROM coches
    WHERE matricula='M3020KY';
    DBMS_OUTPUT.PUT_LINE('Marca ' || v_coche.marca);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No hay coches con esa matricula');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Hay mas de un coche con esa matrícula');
END;
```




TIPOS



Tipos de excepciones

- **Definidas por el servidor Oracle**

- Dos tipos

 - **Con nombre (predefinidas).**

 - **Sin nombre (internamente definidas).**

- Se lanzan implícitamente cuando se produce un error de ejecución.

- También es posible lanzarlas de forma explícita (**RAISE**).

Tipos de excepciones

- **Definidas por el usuario**

- Dos tipos

- **Con nombre (hay que declararlas previamente).**

- **Sin nombre (RAISE_APPLICATION_ERROR).**

- **Se lanzan explícitamente .**

Category	Definer	Has Error Code	Has Name	Raised Implicitly	Raised Explicitly
Internally defined	Runtime system	Always	Only if you assign one	Yes	Optionally ¹
Predefined	Runtime system	Always	Always	Yes	Optionally ¹
User-defined	User	Only if you assign one	Always	No	Always



DEFINIDAS
POR ORACLE

Excepciones Oracle: Con nombre

- Excepciones con nombre definidas por Oracle y asociadas con un error en tiempo de ejecución.
- Son lanzadas implícitamente cuando se produce el error.
- También es posible lanzarlas de forma explícita (**RAISE**).

Nombre	Error Oracle
CURSOR_ALREADY_OPEN	ORA-06511
DUP_VAL_ON_INDEX	ORA-00001
INVALID_CURSOR	ORA-01001
INVALID_NUMBER	ORA-01722
LOGIN_DENIED	ORA-01017
NO_DATA_FOUND	ORA-01403
NOT_LOGGED_ON	ORA-01012
PROGRAM_ERROR	ORA-06501
STORAGE_ERROR	ORA-06500
TIMEOUT_ON_RESOURCE	ORA-00051
TOO_MANY_ROWS	ORA-01422
TRANSACTION_BACKED_OUT	ORA-00061
VALUE_ERROR	ORA-06502
ZERO_DIVIDE	ORA-01476

Excepciones Oracle: Con nombre

EXCEPCIÓN	SE ACTIVAN CUANDO...
CURSOR_ALREADY_OPEN	Intentamos abrir un cursor ya abierto
DUP_VAL_ON_INDEX	Se intenta almacenar un valor que crearía duplicados en una col. con restricción UNIQUE
INVALID_CURSOR	Se intenta hacer una operación no válida sobre un cursor (abrir un cursor abierto, por ej.)
INVALID_NUMBER	Fallo al intentar convertir una cadena a un valor numérico
LOGIN_DENIED	Usuario o clave no válido
NOT_LOGGED_ON	Se intenta acceder a la base de datos sin estar conectado
NO_DATA_FOUND	Una sentencia SELECT no devuelve ninguna fila
PROGRAM_ERROR	Problema interno en la ejecución del programa
STORAGE_ERROR	Error de memoria en la ejecución
TIMEOUT_ON_RESOURCE	Se excede el tiempo de espera para un recurso
TOO_MANY_ROWS	Una sentencia SELECT devuelve más de una fila
VALUE_ERROR	Error aritmético, de conversión o truncamiento
ZERO_DIVIDE	Se intenta dividir entre 0

Excepciones Oracle: Con nombre

```
SET SERVEROUTPUT ON;
DECLARE
    v_coche coches%rowtype;
BEGIN
    SELECT * into v_coche
    FROM coches
    WHERE matricula='M3020KY';
    DBMS_OUTPUT.PUT_LINE('Marca ' || v_coche.marca);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No hay coches con esa matricula');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Hay mas de un coche con esa matrícula');
END;
```

Excepciones Oracle: Con nombre

```
DECLARE
    dato1 NUMBER := 5;
    dato2 NUMBER := 0;
    dato3 NUMBER := 0;
BEGIN
    dato3:=dato1/dato2;
    DBMS_OUTPUT.PUT_LINE(dato3);
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE('División por cero');
        dato3 := NULL;
END;
```


Excepciones Oracle: Con nombre

```
DECLARE
    dato1 NUMBER := 5;
    dato2 NUMBER := 0;
    dato3 NUMBER := 0;
BEGIN
    IF dato2=0 THEN
        RAISE ZERO_DIVIDE;
    ELSE
        dato3:=dato1/dato2;
        DBMS_OUTPUT.PUT_LINE(dato3);
    END IF;
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE('División por cero');
        dato3 := NULL;
END;
```

Excepciones Oracle: Sin nombre

- Excepciones asociadas a un error interno (con un código asociado) pero no un nombre.
- Son lanzadas implícitamente cuando se produce el error.
- Se pueden tratar de dos formas
 - Con la clausula WHEN OTHERS THEN.
 - Asociándoles un nombre

Excepciones Oracle: Sin nombre

- Con la clausula WHEN OTHERS THEN.

```
...  
EXCEPTION  
    ...  
    [WHEN OTHERS THEN]  
        Sentencias;  
END;
```

```
DECLARE  
    v_dni mecanicos.dni%TYPE := '1001';  
BEGIN  
    DELETE FROM mecanicos where DNI = v_dni;  
EXCEPTION  
    WHEN OTHERS THEN  
        DBMS_OUTPUT.PUT_LINE('Error al borrar');  
END;
```

Excepciones Oracle: Sin nombre

- Asociándoles un nombre
 1. Declarar un nombre para la excepción (en la sección de declaraciones)
 2. Asociar el nombre al número de error correspondiente (también en la sección de declaraciones)

```
nombre EXCEPTION;  
PRAGMA EXCEPTION_INIT(nombre, codError) ;
```

3. Capturar la excepción como una excepción con nombre.

Excepciones Oracle: Sin nombre

```
DECLARE
    integridad_referencial EXCEPTION;
    PRAGMA EXCEPTION_INIT(integridad_referencial, -2292);
    v_dni mecanicos.dni%TYPE := '1001';
BEGIN
    DELETE FROM mecanicos where DNI = v_dni;
EXCEPTION
    WHEN integridad_referencial THEN
        DBMS_OUTPUT.PUT_LINE('Error de integridad referencial');
END;
```

Excepciones Oracle: SQLCODE y SQLERRM

- Cuando se produce una excepción hay dos atributos que retornan información sobre ellas
 - **SQLCODE**: Código del error.
 - **SQLERRM**: Mensaje asociado al código de error.

Excepciones Oracle: SQLCODE y SQLERRM

```
DECLARE
    integridad_referencial EXCEPTION;
    PRAGMA EXCEPTION_INIT(integridad_referencial,-2292);
    v_dni mecanicos.dni%TYPE := '1001';
BEGIN
    DELETE FROM mecanicos where DNI = v_dni;
EXCEPTION
    WHEN integridad_referencial THEN
        DBMS_OUTPUT.PUT_LINE(SQLCODE || ': ' || SQLERRM );
END;
```



DEFINIDAS
POR EL USUARIO

Definidas por usuario: Con nombre

- El programador puede crear sus propias excepciones con nombre y usarlas para gestionar mejor los errores en sus programas.

1. Declarar la excepción (en la sección de declaraciones)

```
nombre EXCEPCION;
```

2. Lanzar/activar la excepción (RAISE) (en la sección BEGIN)

```
RAISE nombre;
```

3. Capturar la excepción (en la sección EXCEPTION)

Definidas por usuario: Con nombre

```
DECLARE
    dato1 NUMBER := 5;
    dato2 NUMBER := 0;
    dato3 NUMBER := 0;
    dividir_por_cero EXCEPTION;
BEGIN
    IF dato2 = 0 THEN
        RAISE dividir_por_cero;
    ELSE
        dato3:=dato1/dato2;
        DBMS_OUTPUT.PUT_LINE(dato3);
    END IF;
EXCEPTION
    WHEN dividir_por_cero THEN
        DBMS_OUTPUT.PUT_LINE('División por cero');
        dato3 := NULL;
END;
```

Definidas por usuario: Con nombre

```
CREATE OR REPLACE PROCEDURE subirSueldo (v_dni IN NUMBER,  
v_incremento IN NUMBER) IS  
    v_salario_actual NUMBER(10);  
    salario_nulo EXCEPTION;  
BEGIN  
    SELECT salario INTO v_salario_actual  
    FROM mecanicos  
    WHERE dni=v_dni;  
    IF v_salario_actual IS NULL THEN  
        RAISE salario_nulo;  
    ELSE  
        UPDATE mecanicos SET salario=salario+v_incremento  
        WHERE dni=v_dni;  
    END IF;  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
        DBMS_OUTPUT.PUT_LINE('ERROR: Mecanico no encontrado');  
    WHEN salario_nulo THEN /* tratamiento */  
        DBMS_OUTPUT.PUT_LINE('ERROR: Salario nulo');  
END subirSueldo;
```

Definidas por usuario: Sin nombre

- El paquete DBMS_STANDARD incluye el procedimiento RAISE_APPLICATION_ERROR que permite:
 - Activar errores y definir lanzar excepciones tras lo que terminará la ejecución del bloque.
 - Todas las modificaciones que hubiera hecho el bloque se desharán.
 - Solo se puede llamar desde un subprograma almacenado.
 - Se usa en la sección BEGIN y/o EXCEPTION.

Definidas por usuario: Sin nombre

- Formato:

RAISE_APPLICATION_ERROR(numErr,menErr);

- **numErr:**

- Valor negativo entre -20000 y -20999.

- **menErr**

- Cadena de hasta 512 caracteres.

Definidas por usuario: Sin nombre

```
CREATE OR REPLACE PROCEDURE subirSueldo (v_dni IN NUMBER,  
v_incremento IN NUMBER) IS  
    v_salario_actual NUMBER(10);  
BEGIN  
    SELECT salario INTO v_salario_actual  
    FROM mecanicos  
    WHERE dni=v_dni;  
    IF v_salario_actual IS NULL THEN  
        RAISE_APPLICATION_ERROR(-20000,'El salario actual es nulo');  
    ELSE  
        UPDATE mecanicos SET salario=salario+v_incremento WHERE  
dni=v_dni;  
    END IF;  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
        RAISE_APPLICATION_ERROR(-20001,'No existe el mecánico');  
END subirSueldo;
```

Definidas por usuario: Sin nombre

```
DECLARE
    e_salario_nulo EXCEPTION;
    e_no_existe_mecanico EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_salario_nulo,-20000);
    PRAGMA EXCEPTION_INIT(e_no_existe_mecanico,-20001);
BEGIN
    subirSueldo ('10030', 1000);
    DBMS_OUTPUT.PUT_LINE('Se ha subido el salario');
EXCEPTION
    WHEN e_no_existe_mecanico THEN
        DBMS_OUTPUT.PUT_LINE('ERROR: No existe el mecanico');
    WHEN e_salario_nulo THEN
        DBMS_OUTPUT.PUT_LINE('ERROR: Salario nulo');
END;
```



PROPAGACIÓN DE EXCEPCIONES



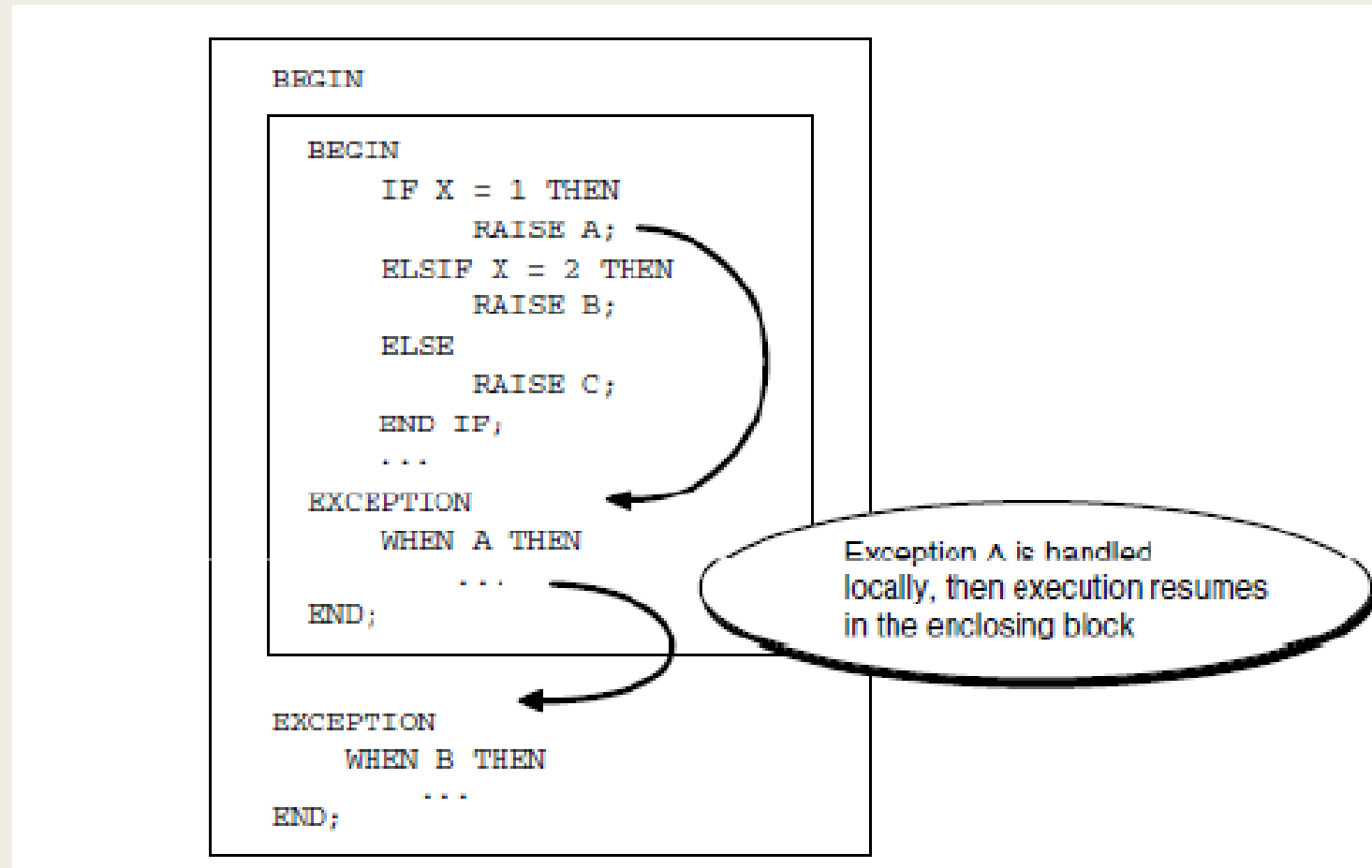
Propagación de excepciones

Cuando se produce una excepción:

- El control del programa se pasa a la zona de excepciones del bloque actual.
 - Si la excepción en cuestión no está tratada en ella, la excepción se propaga al bloque que llamó al bloque actual y así sucesivamente hasta que se encuentre tratamiento para la excepción.
 - Si no se encuentra tratamiento para la excepción el programa terminará con error.
- **Una vez tratada la excepción en un bloque**, se devuelve el control al bloque que llamó al que trató la excepción, con independencia del que la disparó.

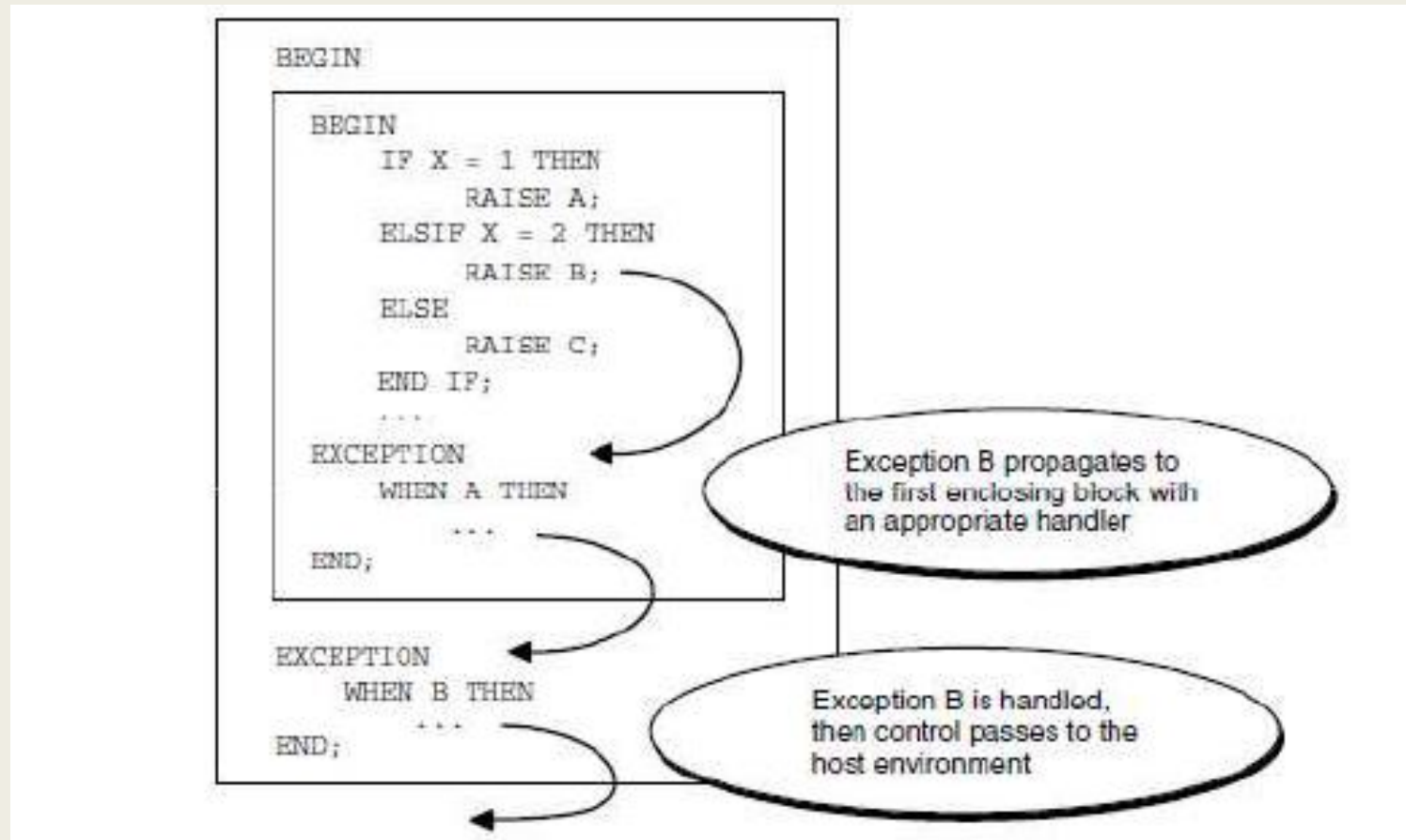
Propagación de excepciones

Ejemplo 1: excepción que no se propaga



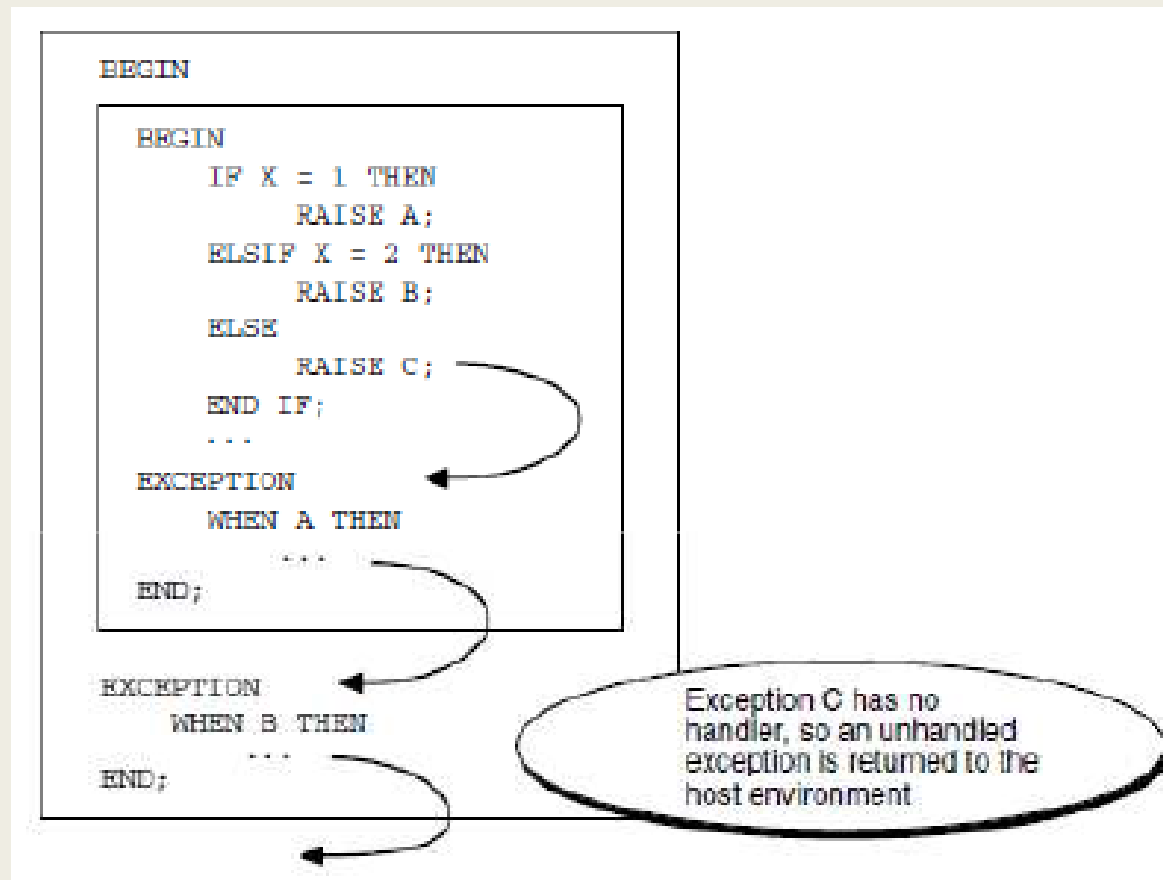
Propagación de excepciones

- **Ejemplo 2: excepción que se propaga y es gestionada en otro bloque.**



Propagación de excepciones

- **Ejemplo 3: excepción que se propaga y no es tratada en ningún bloque.**



Propagación de excepciones

- La cláusula **WHEN OTHERS** siempre tratará las excepciones que no aparezcan consideradas en las cláusulas WHEN previas.
- Es una manera de evitar que las excepciones se propaguen al bloque llamante.
- **Por lo tanto cuando se produce una excepción se puede:**
 - Capturar la excepción en la sección **EXCEPTION** del **mismo bloque** (con las cláusulas **WHEN**).
 - Capturar la excepción en bloques que llaman al bloque donde se produce teniendo en cuenta que las excepciones se propagan.

Excepciones en la declaración

- Una excepción **que se activa en la sección de declaraciones**
 - Ej.: fallo en la inicialización de una variable) se **propagará automáticamente al bloque llamante independientemente de que esté tratada o no en el bloque actual.**

Excepciones en manejadores

- Se puede activar una excepción dentro de la sección EXCEPTION de cualquier bloque, bien voluntariamente, o bien por un error (al tratar la excepción) que la activa.
- La excepción **se propagará automáticamente al bloque llamante** sin comprobar si existe tratamiento para la nueva excepción en el bloque en el que se activo.
- La excepción original se perderá ya que dos excepciones no pueden estar activas al mismo tiempo.

Continuar en la instrucción siguiente

- Si después de tratar una excepción se quiere volver a la instrucción siguiente a la que activó la excepción:
 - Hay que colocar la sentencia o sentencias que nos pueden activar la excepción dentro de un subbloque que a su vez trate la posible excepción.

```
Sentencia1;  
  BEGIN  
    SELECT INTO .... /* puede activar NO_DATA_FOUND */  
  EXCEPTION  
    WHEN NO_DATA_FOUND THEN ...  
  END;  
Sentencia2;
```


Excepciones no tratadas

- Si no existe un manejador (WHEN) para tratar una excepción se envía un mensaje de excepción no tratada al entorno del host que invoca el programa.
- Si la excepción no se trata en funciones/procedimientos
 - Los parámetros OUT e IN OUT mantiene el valor que tuviesen antes de invocar a la/el función/procedimiento.
 - PL/SQL no realiza ROLLBACK de los cambios realizados por la/el función/procedimiento