

# Introducción a PL/SQL

ANEXO Tema 4

Bases de Datos  
Curso 2021/22

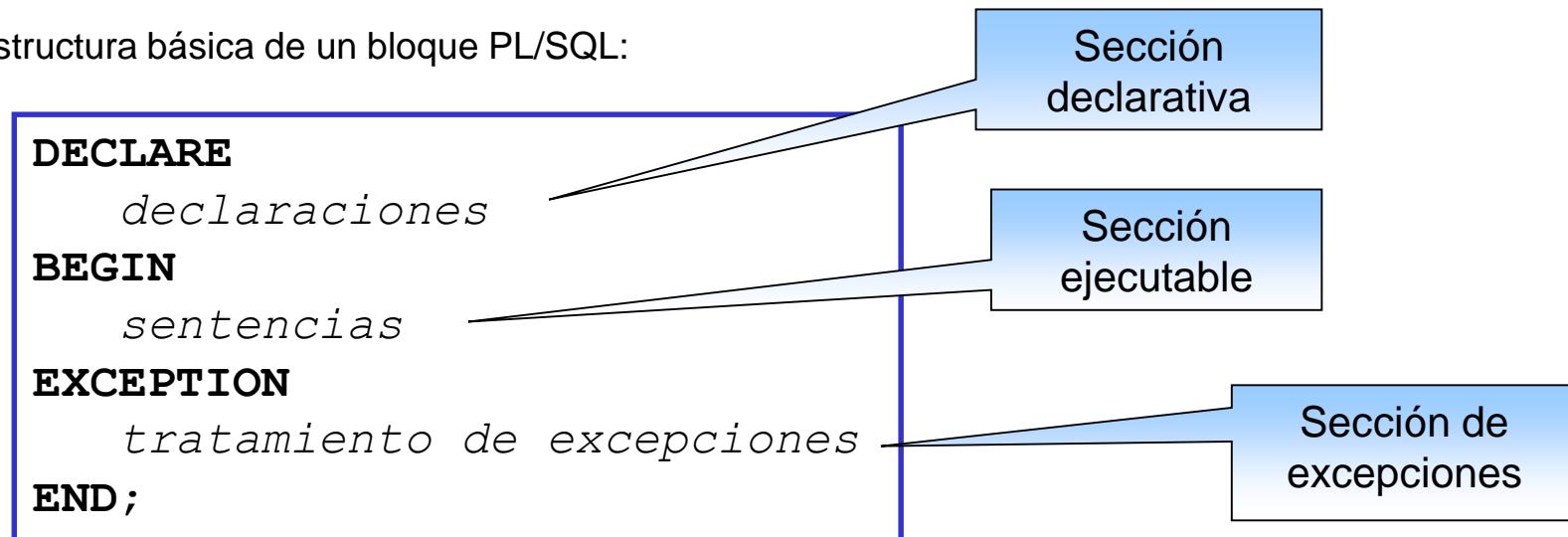
# Índice

---

1. Introducción
2. Ejecución de PL/SQL
3. Variables, constantes y tipos
4. Estructuras de control de flujo
5. SELECT ... INTO
6. Gestión de excepciones

# Introducción

- 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:



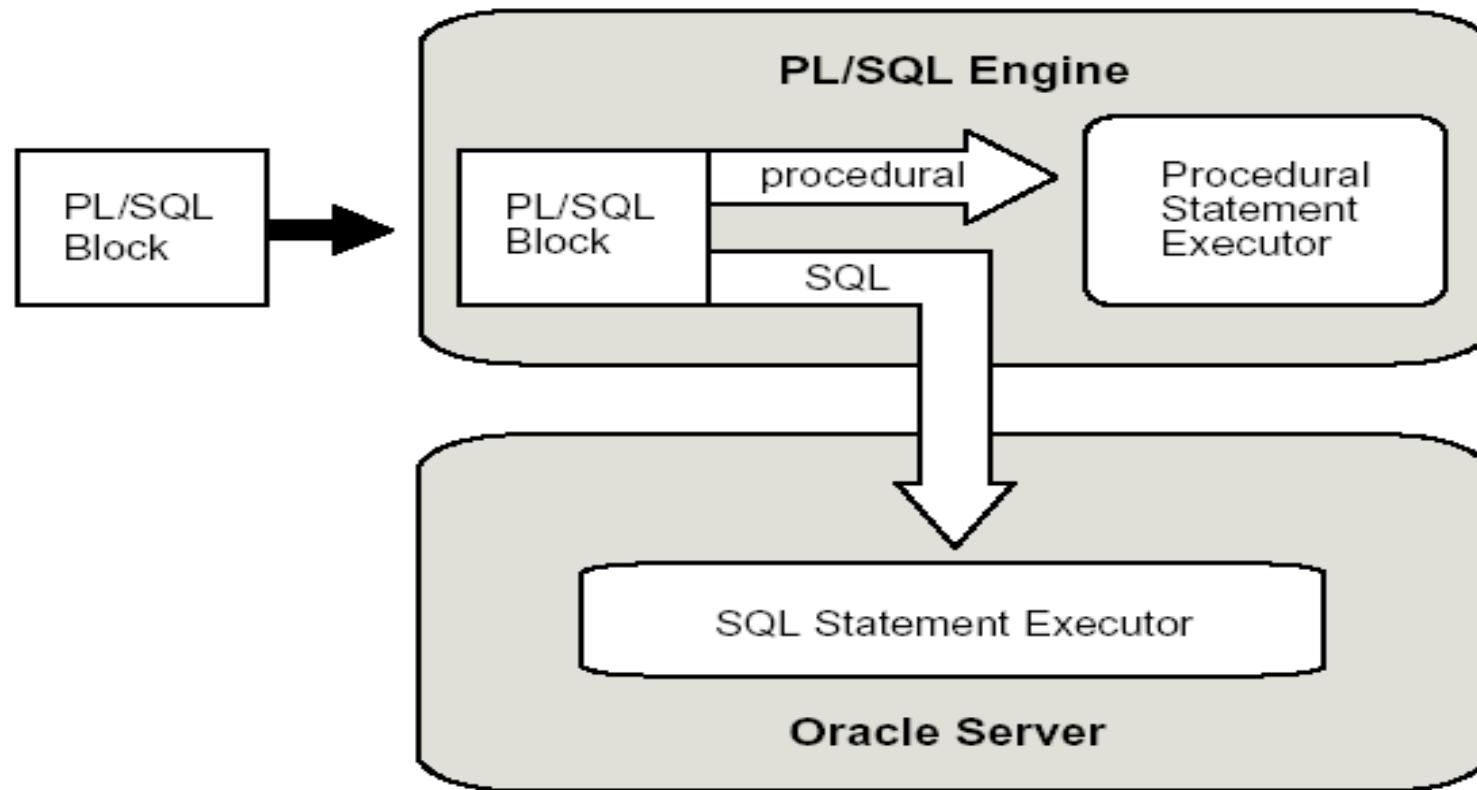
# Ejecución de PL/SQL (I)

- 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);
```

# Ejecución de PL/SQL (II)



# Variables, constantes y tipos (I)

- 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;
```

# Variables, constantes y tipos (II)

- 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];
```

# Variables, constantes y tipos (III)

- 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;
```

# Estructuras de control de flujo (I)

## □ 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;
```

# Estructuras de control de flujo (II)

- ❑ 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]
```

# SELECT ... INTO (I)

- 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**
  - **NO puede devolver ninguna fila**

Al usar **SELECT ... INTO** → la consulta debe devolver una única tupla

# SELECT ... INTO (II)

## 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;
```

# Gestión de excepciones (I)

- 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;
```

# Gestión de excepciones (II)

## 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;
```

# Gestión de excepciones (III)

**Ejemplo:** Control del número de 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;
    ...

```

# Gestión de excepciones (IV)

**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;
```

# Gestión de excepciones (V)

- 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

# Gestión de excepciones (VI)

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;
```