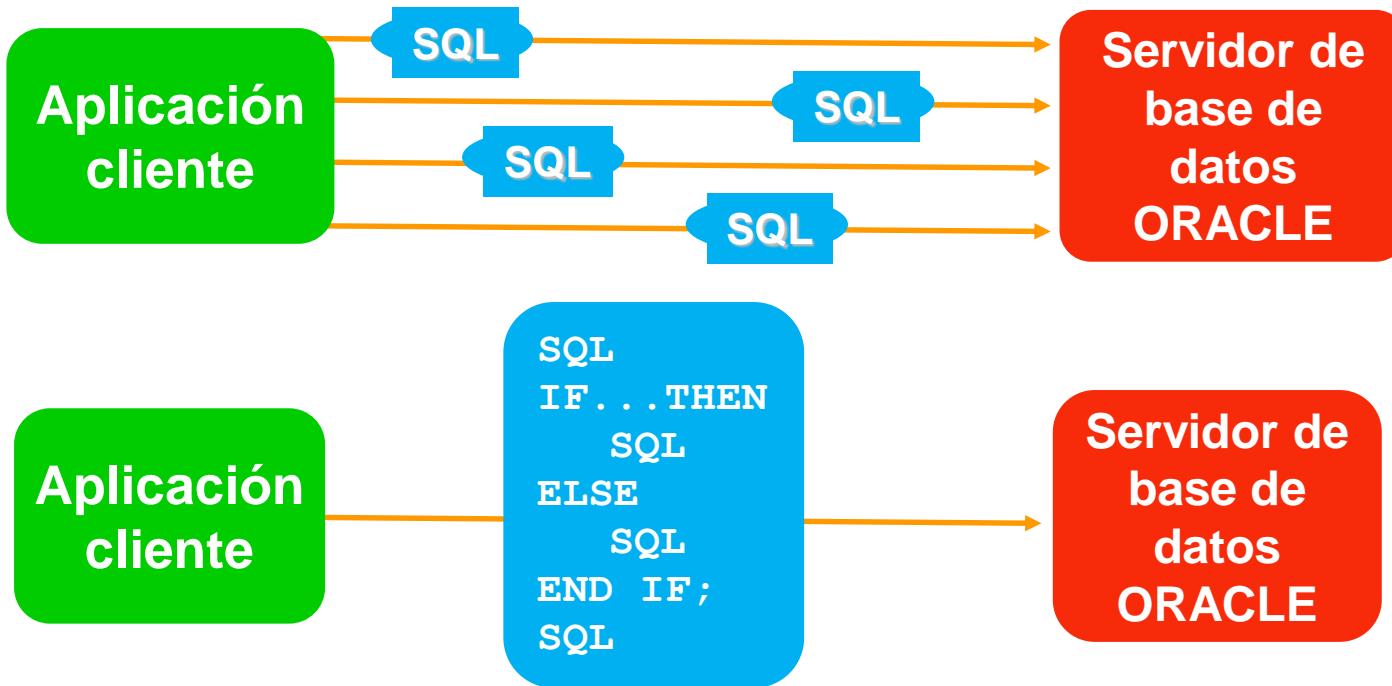


# INTRODUCCIÓN AL PL-SQL

# QUE ES PL/SQL?

- El PL/SQL (Procedural Language/SQL) es, como su nombre lo indica, un lenguaje que amplía la funcionalidad de SQL añadiendo estructuras que también pueden encontrarse en otros lenguajes procedimentales de tercera generación.
- La manipulación de los datos y declaraciones de los queries son incluidos dentro de las unidades de código procedimentales

# Performance del PL/SQL



Particularmente en un modelo cliente/servidor que utiliza una aplicación con simples consultas SQL, la máquina cliente envía una solicitud al servidor por cada consulta SQL generando múltiples comunicaciones por la red. En cambio, si todas las sentencias son contenidas en un único bloque PL/SQL, éste viaja al servidor como una sola petición, reduciendo el tráfico de la red y aumentando la velocidad de la aplicación.

# Beneficios de PL/SQL

- Integración de construcciones procedimentales con SQL
- Mejora de la performance
- Desarrollo modularizado de aplicaciones
- Integración con otras herramientas de ORACLE
- Portabilidad
- Manejo de excepciones

# Cada bloque SQL tiene las siguientes partes:

Sección	Descripción	Inclusión
Declarativa	Contiene todas las variables, constantes, cursores que serán referenciadas en la sección ejecutable	Opcional
Ejecutable	Contiene sentencias SQL que manipulan los datos de la Base y del bloque	Obligatorio
Manejo de errores	Especifica las acciones que se deben ejecutar cuando existe una condición anormal	Opcional

# Estructura del bloque PL/SQL

- **DECLARE** – Opcional
  - Variables, constantes, cursor
  - Excepciones definidas por el usuario
- **BEGIN** – Obligatorio
  - Sentencias SQL
  - Sentencias PL/SQL de control
- **EXCEPTION** – Opcional
  - Acciones que se realizan cuando ocurre un error
- **END;** – obligatorio

# Ejemplo de PL/SQL anónimo

```
DECLARE
    v_articulo_id    b_articulos.id%TYPE;
BEGIN
    SELECT id INTO v_articulo_id FROM b_articulos
    WHERE id = &v_articulo_id;
    DELETE FROM prueba.b_detalle_ventas WHERE
        id_articulo = v_articulo_id;
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        INSERT INTO tabla_excepcion (mensaje) VALUES
            ('ocurrió un error en la BD');
        COMMIT;
END;
```

# Tipo de Bloques

Anónimo

```
[DECLARE]  
  
BEGIN  
    --sentencias  
  
[EXCEPTION]  
  
END ;
```

Procedimiento

```
PROCEDURE nombre  
IS  
  
BEGIN  
    --sentencias  
  
[EXCEPTION]  
  
END ;
```

Función

```
FUNCTION nombre  
RETURN  
tipo_dato  
IS  
BEGIN  
    --sentencias  
    RETURN value;  
[EXCEPTION]  
  
END ;
```

# Qué son las Unidades de Programas de PL/SQL

- Son bloques de PL/SQL nombrados
- Existen tres categorías principales:
  - **Procedimientos** para realizar acciones
  - **Funciones** para calcular un valor
  - **Paquetes** que agrupan procedimientos y funciones lógicamente relacionados
- Son guardados en la BD o en la aplicación

# Componentes del Subprograma

## **CABECERA – opcional**

- Nombre del subprograma, tipo y argumentos

## **SECCION DECLARATIVA – opcional**

- Identificadores locales

## **SECCION EJECUTABLE – obligatorio**

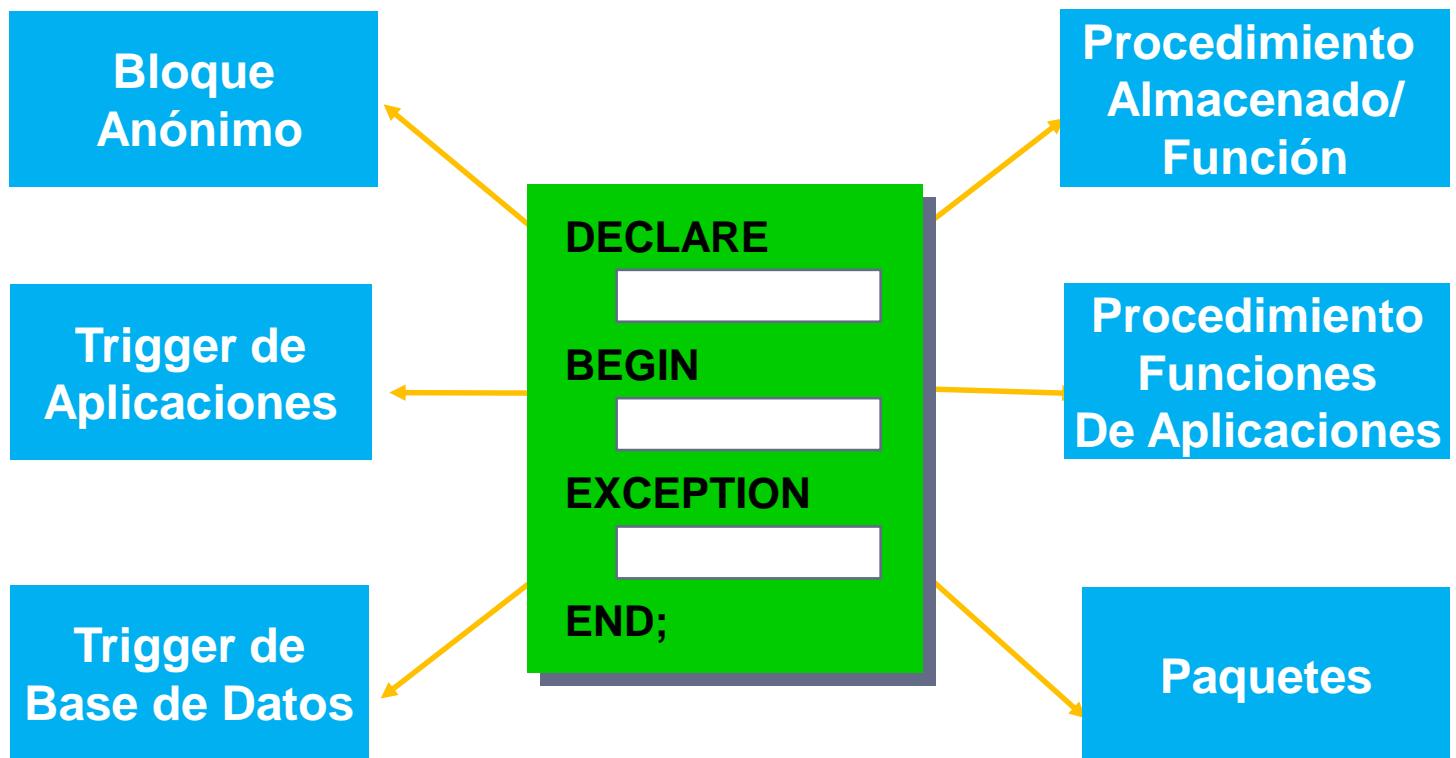
- Sentencias SQL
- Sentencias de control PL/SQL

## **MANEJO DE EXEPICION – opcional**

- Acciones que se ejecutan cuando hay error

## **FINAL (END); – obligatorio**

# Construcción de Programas



# Para ver la salida en consola:

```
SQL> SET SERVEROUTPUT ON  
...  
DBMS_OUTPUT.PUT_LINE('El id: ' ||  
    to_char(v_id, '99'));
```

# Declarando Variables y Constantes: Sintaxis

```
identificador [CONSTANT] tipo_dato [NOT NULL]  
[ := | DEFAULT expr] ;
```

- Pautas
  - Nombre las variables de acuerdo a las convenciones
  - Inicialice las constantes y variables designadas como NOT NULL
  - Inicialice identificadores usando al operador de la asignación (:=) o por la palabra reservada DEFAULT.
  - Declare un identificador por línea

# Tipos de variables admitidas

- Tipos
  - BINARY\_INTEGER
  - NUMBER [(número, precisión)]
  - CHAR [(longitud máxima)]
  - LONG
  - VARCHAR2(máxima longitud)
  - DATE
  - BOOLEAN

# Declaración y asignación de variables

```
v_genero           CHAR(1);  
v_contador         BINARY_INTEGER := 0;  
v_total_sal        NUMBER(9,2)  := 0;  
v_fecha_asiento   DATE := SYSDATE + 7;  
v_fecha_inicio    DATE DEFAULT SYSDATE;  
c_tasa             CONSTANT NUMBER(3,2) := 8.25;  
v_valido           BOOLEAN NOT NULL := TRUE;  
ciudad             VARCHAR2(20) := 'Luque';
```

# Utilización de un carácter alternativo para expresiones de tipo VARCHAR

- Cuando una expresión alfanumérica debe apóstrofe, entonces tenemos las siguientes alternativas:
  - Utilizar el signo ' de forma doble
  - Definir un carácter alternativo para encerrar las expresiones alfanuméricas

## Utilización de un carácter alternativo para expresiones de tipo VARCHAR - Ejemplo

```
personaje1 := 'Scarlett O"hara';
```

ó bien

```
personaje1 := q'[Scarlett O'hara]';
```

# El Atributo %TYPE

- Permite declarar una variable a partir de:
  - Otra variable declarada previamente
  - Una definición de columna de una base de datos.
- Declarando la variable con %type permite que el PL/SQL determine el tipo de dato y tamaño de la variable.

# El Atributo %TYPE: Ejemplos

```
v_apellido          b_empleados.apellido%TYPE;  
v_nombre            b_empleados.nombre%TYPE;  
v_balance           NUMBER(7,2);  
v_balance_minimo   v_balance%TYPE := 10;
```

## Ventajas de usar el atributo de %TYPE

- Los tipos de datos de la columna de la base de datos subyacente pueden ser desconocidos
- Los tipos de datos de la columna de la base de datos subyacente pueden cambiar al ejecutarse

# Reglas de sintaxis de bloques PL/SQL

- Una sentencia puede continuar en varias líneas
- Los nombres de variables deben empezar con un carácter alfabético y pueden tener hasta 30 caracteres de longitud
- Se pueden poner comentarios:
  - /\*                \*/ (para varias líneas)
  - -- (para una sola línea)
- Los bloques pueden ser anidados

# Guías de programación

- Documentar el código con comentarios
- Desarrollar una convención para la denominación de variables
- Facilitar la lectura del código utilizando indentación
- Desarrollar una convención para la codificación. Ej:
  - Mayúsculas para palabras claves de SQL y PL/SQL
  - Minúsculas para variables, tablas, columnas, etc.

# ALCANCE DE LAS VARIABLES

....

**x BINARY\_INTEGER**

**BEGIN** —————

....

**Alcance de x**

**DECLARE** —————

**y NUMBER;**

**alcance de y**

**BEGIN** —————

.....

**END;**

.....

**END;** —————

# Operadores en PL/SQL

- Lógicos
- Aritméticos
- Concatenación
- Paréntesis para control de secuencia de ejecución de las operaciones
- Exponente (\*\*)

Igual  
Que en  
SQL

# Operadores en PL/SQL (Ejemplos)

- **Incrementar un índice:**

```
v_count := v_count + 1;
```

- **Setear una bandera (una variable definida como boolean)**

```
Es igual := (v_n1 = v_n2);
```

- **Validar si una columna tiene valor nulo:**

```
v_valido := (emp_id is NOT NULL);
```

- **Asignando valores a un alfanumérico:**

```
v_nombre := emp.nombre || ' ' || emp.last_nombre;
```

# CONTROL DE FLUJOS

# Controlando con PL/SQL el flujo de Ejecución

- Se puede cambiar el flujo lógico de las declaraciones utilizando estructuras de control:
  - **Control condicional**
    - Declaración IF, expresión CASE.
  - **Estructuras de control iterativo:**
    - Uso del LOOP:
      - LOOP básico
      - FOR LOOP
      - LOOP WHILE
      - Declaración EXIT

# Declaración IF : Sintaxis

```
IF condición THEN  
    sentencias;  
    [ELSIF condición THEN  
        sentencias;]  
    [ELSE  
        sentencias;]  
END IF;
```

Verificación de condición con la construcción IF:

- ELSIF es una palabra (ELSE IF....)
- END IF finaliza la sentencia.
- Sólo puede existir una cláusula ELSE

# Declaración IF Simple: Ejemplo

```
 . . .
IF apellido = 'Dumas' THEN
    v_job := 'Representante de ventas';
    v_region_id := 35;
END IF;
. . .
```

```
 . . .
IF v_fecha_envio - v_fecha_orden < 5 THEN
    v_bandera := 'Aceptable';
ELSE
    v_bandera := 'Inaceptable';
END IF;
```

# Declaración IF-THEN-ELSIF : Ejemplo

```
    . . .
    IF v_start > 100 THEN
        RETURN (2 * v_start);
    ELSIF v_start >= 50 THEN
        RETURN (.5 * v_start);
    ELSE
        RETURN (.1 * v_start);
    END IF;
    . . .
```

# Expresión CASE

## Sintaxis I

```
CASE variable_seleccion  
    WHEN valor1 THEN  
        expresion1;  
    WHEN  valor2 THEN  
        expresion2;  
    ELSE  
        expresion3;  
END CASE;
```

# Expresión CASE

## Ejemplo Sintaxis I

```
CASE v_rango
    WHEN 1    THEN
        v_comision := v_total*0.10;
    WHEN 2    THEN
        v_comision :=  v_total*0.5;
    ELSE
        v_comision := v_total;
END CASE;
RETURN v_comision;
```

# Expresión CASE

## Sintaxis 2

```
CASE  
    WHEN condicion1 THEN  
        expresion1;  
    WHEN condicion2 THEN  
        expresion2;  
    ELSE  
        expresion3;  
END CASE;
```

# Expresión CASE

## Ejemplo Sintaxis 2

```
CASE
```

```
WHEN (v_cal between 0 and 10) THEN  
      v_comision := v_total*0.10;
```

```
WHEN (v_cal between 11 and 20) THEN  
      v_comision := v_total*0.5;
```

```
ELSE
```

```
      v_comision := v_total ;
```

```
END CASE ;
```

# Sentencia GOTO

Transfiere el flujo de ejecución a una etiqueta definida en el cuerpo del programa.

Las etiquetas se declaran del siguiente modo: <<NOMBRE\_ETIQUETA>>

```
CASE

    WHEN V_MONTO_VENTAS BETWEEN 1 AND 10000000 THEN
        V_PORC:=0.05;
        GOTO PASO1;

    WHEN V_MONTO_VENTAS>10000000 THEN
        V_PORC:=0.1;
        GOTO PASO1;

    ELSE
        V_COMISION:=0;
        GOTO PASO2;

END CASE;

<<PASO1>>

    V_COMISION:=V_MONTO_VENTAS*V_PORC;

<<PASO2>>

    DBMS_OUTPUT.PUT_LINE('COMISION: '||V_COMISION);

END;
```

# Consideraciones Sentencia GOTO

Una instrucción GOTO no puede transferir el control:

- A una instrucción IF , una instrucción CASE , una instrucción LOOP o un sub-bloque.
- De una cláusula de la instrucción IF a otra, o de una cláusula de la instrucción CASE a otra.
- De un subprograma.
- A un controlador de excepciones.
- De un controlador de excepciones al bloque actual.

# Construcción de Condiciones Lógicas

- Se puede manejar valores nulos con el operador IS NULL
- Cualquier expresión que contiene un valor nulo evalúa a NULO
- Las expresiones encadenadas con valores nulos tratan valores nulos como un string vacío

# Tabla Lógica

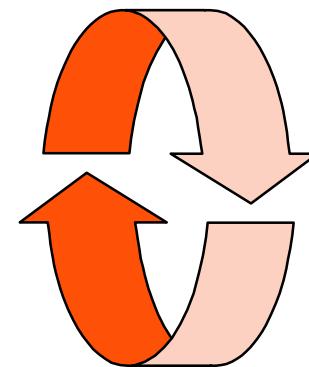
AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

# Declaraciones LOOP

- LOOP repite una sentencia o sucesión de sentencias múltiples veces.
- Tres tipos de LOOP:
  - LOOP básico
  - FOR ...LOOP
  - LOOP ...WHILE



# Loop básico: Sintaxis

- Realiza iteraciones de una sentencia
- Sin la declaración EXIT, el LOOP sería infinito.

```
LOOP  
    sentencia 1;  
    . . .  
    EXIT [WHEN condición];  
END LOOP;
```

# Loop Básico - Ejemplo

- Inserte registros para los 12 meses del año 2012 en la tabla b\_liquidacion

```
DECLARE
    v_anio          b_liquidacion.anio%TYPE := 2012;
    v_mes           NUMBER(2)   := 1;
    v_contador      NUMBER(2)   := 1;

BEGIN
    .
    .
    .
    LOOP
        INSERT INTO b_liquidacion (fecha_liquidacion, anio, mes, id )
        VALUES (sysdate, v_anio, v_mes, v_contador );
        v_mes := v_mes + 1;
        v_contador := v_contador + 1;
        EXIT WHEN v_contador > 12;
    END LOOP;
    .
    .
```

# FOR.. LOOP

- Con el FOR ...LOOP no hace falta evaluar una condición

```
FOR indice IN [REVERSE]
    valor_inferior.. Valor_superior LOOP
        sentencia 1;
        sentencia 2;
    END LOOP;
```

**La variable que hace de índice no se declara,  
ya que el LOOP la define de modo  
automático**

# For Loop

- Ejemplo de uso del FOR ...LOOP

```
FOR  i  IN 1 ..10 LOOP
    v_contador  := v_contador + 1;
    v_contador2 := i;
END LOOP;
```

# While ... Loop

- Repite una serie de sentencias entretanto la condición sea verdadera

```
WHILE condición LOOP  
    sentencia 1;  
    sentencia 2;  
END LOOP;
```

La condición se evalúa antes de entrar a iterar

# While Loop

- Ejemplo: inserte las 10 primeras líneas para Marzo/2012 en el mayor

```
DECLARE
    v_anio  b_mayor.anio%TYPE := 2012;
    v_mes   b_mayor.mes%TYPE := 3;
    v_contador NUMBER(2) := 1;

BEGIN
    . . .
    WHILE v_contador <= 10 LOOP
        INSERT INTO b_mayor (id_mayor, anio, mes )
            VALUES (v_contador, v_anio, v_mes);
        v_contador := v_contador + 1;
    END LOOP;
```

# Recuperación de Datos en PL/SQL

# Recuperación de datos con la sentencia SELECT

- Se utiliza la cláusula SELECT para recuperar datos de la Base de Datos. A diferencia del SQL sin embargo, el PL/SQL añade una cláusula adicional obligatoria: INTO
- La cláusula INTO lista las variables que recibirán los datos a partir del SELECT
- Cuando la cláusula SELECT se incluye en PL/SQL debe recuperar exactamente una fila, pues en caso contrario provocará un error

# Recuperación de datos con la sentencia SELECT: sintaxis

```
SELECT      lista de columnas
INTO        lista_variables | nombre_Registro
FROM        tabla
WHERE       condición;
```

- Cada sentencia debe terminar con el punto y coma (;)
- No se puede obviar la cláusula INTO en el SELECT si se incluye en PL/SQL
- Las variables que siguen a la cláusula INTO deben ser del mismo número y tipo que las que forman parte del SELECT.
- Para asegurarse que las variables receptoras son del mismo tipo que las columnas seleccionadas, puede definir aquellas con el atributo %TYPE

# Recuperación de datos con la sentencia SELECT: Ejemplo

- Recupere el nombre y la asignación para la categoría especificada

```
DECLARE
  v_cat      b_categorias_salariales.cod_categoria%type :=56;
  v_nom      b_categorias_salariales.nombre_cat%type;
  v_asig     b_categorias_salariales.asignacion%type;
BEGIN
  select nombre_cat, asignacion into v_nom, v_asig
    from b_categorias_salariales
   where cod_categoria = :v_cat;
END;
```

# Recuperación de datos con la sentencia SELECT: sintaxis

- Utilización de una variable tipo “registro” para recuperar de una sola vez todas las columnas en un SELECT.

```
DECLARE
    reg_area  b_areas%ROWTYPE;
BEGIN
    SELECT  *
        INTO  reg_area    --Registro PL/SQL
        FROM  b_areas
        WHERE  id = &v_dept_id;
END;
```

# Convenciones

- Use una convención para evitar ambigüedad en la cláusula WHERE
- Las columnas de la base de datos e identificadores deben tener nombres distintos
- Si se usan columnas con el mismo nombre de la tabla, pueden surgir errores de sintaxis, puesto que PL/SQL busca primero el nombre de la columna en la tabla

# Ejemplo de errores

- Recupere el nombre y la asignación para la categoría especificada

```
DECLARE
  v_cat      b_categorias_salariales.cod_categoria%type;
  v_nom      b_categorias_salariales.nombre_cat%type;
  v_asig     b_categorias_salariales.asignacion%type;
BEGIN
  select nombre_cat, asignacion into v_nom, v_asig
    from b_categorias_salariales
   where cod_categoria = 834;
END;
```



No existe la categoría

# Manejo de Errores (Excepciones)

- Como se mencionaba, el SELECT debe recuperar una única fila cuando se incluye en PL/SQL
- Se genera un error cuando no se recupera ninguna fila, o cuando se recuperan más de una fila

Condición	Excepción
La sentencia SELECT recupera más de una fila	TOO_MANY_ROWS (Error de ORACLE ORA-01422 )
La sentencia SELECT no recupera ninguna fila	NO_DATA_FOUND (Error de ORACLE ORA-01403)

# Ejemplo de excepción TOO\_MANY\_ROWS

- Recupere el nombre y apellido del empleado que depende del jefe cuya cedula es 952160

```
DECLARE
    v_jefe      b_empleados.cedula_jefe%type:=952160;
    v_nom       b_empleados.nombre%type;
    v_ape       b_empleados.apellido%type;
BEGIN
    select nombre, apellido into v_nom, v_ape
        from b_empleados
    where cedula_jefe = v_jefe;
    --TOO_MANY_ROWS
END;
```



# Manipulación de Datos

- Las sentencias DML se usan de manera convencional, al igual que en SQL:
  - INSERT
  - UPDATE
  - DELETE