

1

Visión General de PL/SQL

Bases de datos II

Programa de Ingeniería de sistemas

Universidad del Cauca

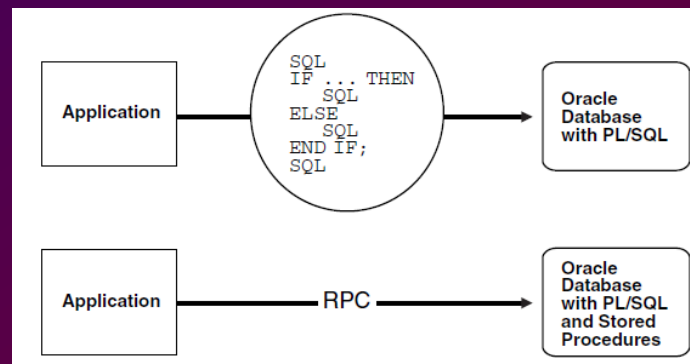
Ing. Wilson Ortega

Introducción (I)

- PL/SQL -Procedural Language/Structured Query Language.
- Lenguaje procedimental que amplía la funcionalidad de SQL añadiendo estructuras habituales en otros lenguajes: variables y tipos, estructuras de control, procedimientos y funciones, paquetes.
- Los procedimientos, funciones, disparadores y paquetes creados con el PL/SQL se almacenan en base de datos.
- Están incluidos dentro de las políticas de seguridad de Oracle y son altamente recomendables, para el tratamiento de datos.

Ventajas PL/SQL (I)

- **Integración con SQL**
 - Consultas, DML, funciones, operadores, alias
 - Soporte de los tipos de datos SQL
- **Alto rendimiento**
 - Es posible enviar un bloque de sentencias a la BD
 - Subprogramas compilados una vez y almacenados como ejecutables



Ventajas PL/SQL (II)

- **Portabilidad**

- Las aplicaciones PL/SQL se pueden ejecutar en cualquier plataforma donde se esté ejecutando la BD

- **Paquetes predefinidos**

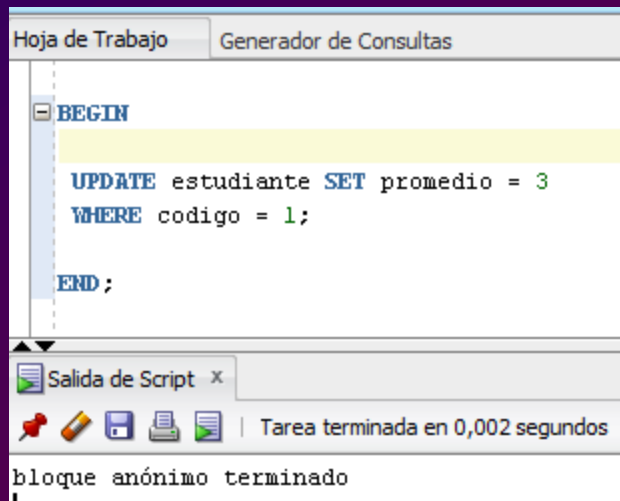
- APIs que se pueden invocar para ejecutar tareas útiles como DBMS_FILE para leer y escribir en archivos o DBMS_OUTPUT para mostrar información en la consola.

Bloques

- La unidad básica en PL/SQL es el bloque.
- Todos los programas PL/SQL están compuestos por bloques, que pueden definirse de forma secuencial o estar anidados.
- Normalmente cada bloque realiza una unidad lógica de trabajo en el programa, separando así unas tareas de otras

Tipos de bloques (I)

- Anónimos (Anonymous blocks):
 - Se construyen de forma dinámica y se ejecutan una sola vez.
 - No se guardan en la BD.
 - Es posible asignarles un label para identificarlos pero aún así se siguen considerando anónimos



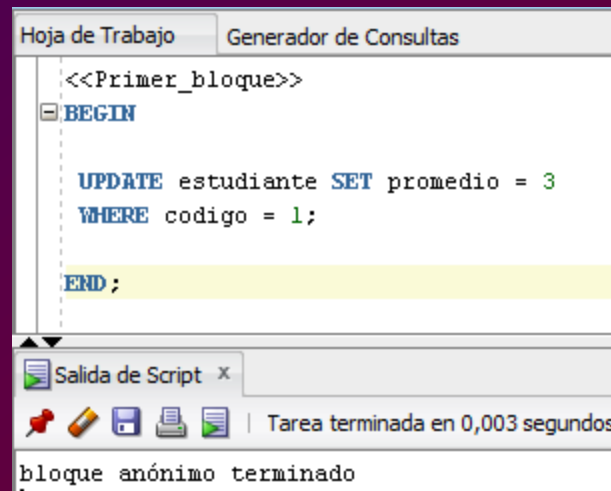
Hoja de Trabajo | Generador de Consultas

```
BEGIN  
  
UPDATE estudiante SET promedio = 3  
WHERE codigo = 1;  
  
END;
```

Salida de Script x

Tarea terminada en 0,002 segundos

bloque anónimo terminado



Hoja de Trabajo | Generador de Consultas

```
<<Primer_bloque>>  
BEGIN  
  
UPDATE estudiante SET promedio = 3  
WHERE codigo = 1;  
  
END;
```

Salida de Script x

Tarea terminada en 0,003 segundos

bloque anónimo terminado

Tipos de bloques (II)

- Con nombre (Named blocks):
 - Se llaman también subprogramas.
 - Se guardan en la BD.
 - Pueden ser procedimientos, funciones o paquetes

```
CREATE OR REPLACE PROCEDURE AsignarPromedio
(
  p_cod NUMBER,
  p_prom NUMBER
)
IS
BEGIN
  UPDATE estudiante SET promedio = p_prom
  WHERE codigo = p_cod;
END ;
```


Tipos de Bloques (III)

Anónimo: **[DECLARE]**
 BEGIN
 sentencias ejecutables;
 [EXCEPTION]
 END;

Procedimiento: **CREATE PROCEDURE** nombre_procedimiento [(parámetros)]
 IS
 [declaración - opcional]
 BEGIN
 sentencias ejecutables;
 [EXCEPTION]
 END;

Función: **CREATE FUNCTION** nombre_funcion [(parámetros)]
 RETURN tipo de dato a devolver
 IS [declaración - opcional]
 BEGIN
 sentencias ejecutables;
 RETURN value;
 [EXCEPTION]
 END;

Identificadores

- Se emplean para dar nombre a los objetos PL/SQL, tales como variables, cursores, tipos y subprogramas.
- Los identificadores constan de una letra, seguida por una secuencia opcional de caracteres, que pueden incluir letras, números, signos de dólar (\$), caracteres de subrayado y símbolos de almohadilla (#). Los demás caracteres no pueden emplearse.
- La longitud máxima es de 30 caracteres
- No diferencia entre mayúsculas y minúsculas
- No usar palabras reservadas

Literales (I)

- **Carácter**
 - Constan de uno o más caracteres delimitados por comillas simples. Se pueden asignar a variables de tipo CHAR o VARCHAR2, sin tener que hacer ningún tipo de conversión: '12345' '100%'
- **Numérico**
 - Representa un valor entero o real, puede asignarse a una variable de tipo NUMBER sin tener que efectuar conversión alguna.
 - Los literales enteros consisten de una serie de dígitos, precedidos opcionalmente por un signo (+ o -). No se permite utilizar un punto decimal en un literal entero. 123 +7 -9

Literales (II)

- Un literal real consta de signo, opcional, y una serie de dígitos que contiene punto decimal. También pueden escribirse utilizando notación científica. -17.7 23.0 1.345E7 -7.12e+12
- Boolean
 - Los literales booleanos representan la verdad o falsedad de una condición y se utilizan en las órdenes IF y LOOP, solo existen tres posibles literales booleanos:
 - TRUE Verdadero
 - FALSE Falso
 - NULL Nulo

Entrada / Salida (I)

- Para mostrar un valor por pantalla:
 - DBMS_OUTPUT.PUT_LINE(cadena);
- Es necesario activar la opción SERVEROUTPUT mediante la siguiente instrucción:
 - SET SERVEROUTPUT ON

```
SET SERVEROUTPUT ON
DECLARE
  v_num NUMBER := 4;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Imprimiendo...');
  DBMS_OUTPUT.PUT_LINE(v_num);
END;
```

Entrada / Salida (II)

- Para leer valores por pantalla se puede usar el comando ACCEPT o las variables de sustitución:
- **Variable de sustitución:** pueden aparecer directamente en la sentencia SELECT sin necesidad de definirla, anteponiendo el símbolo & y SQL nos preguntará el valor que queremos asignarle. Se puede usar SET VERIFY OFF para que no se muestre automáticamente por pantalla el valor anterior de la variable

Entrada / Salida (II)

```
SET SERVEROUTPUT ON
SET VERIFY OFF
```

```
DECLARE
```

```
BEGIN
```

```
    UPDATE estudiante SET promedio =    &v_prom
```

```
    WHERE codigo = &v_cod;
```

```
END;
```

```
SET SERVEROUTPUT ON
SET VERIFY OFF
```

```
DECLARE
```

```
v_codigo NUMBER := &cod;
```

```
BEGIN
```

```
    UPDATE estudiante SET promedio = 5
```

```
    WHERE codigo = v_codigo;
```

```
END;
```

Entrada / Salida (III)

- ACCEPT permite declarar una variable y leer su valor poniendo un mensaje en el Prompt.
- Para utilizar la variable accedemos a ella anteponiéndole el símbolo &.
- **PERO:** No podemos utilizar ACCEPT para leer variables dentro de un bloque PL/SQL, si queremos utilizarlo debemos hacerlo fuera.

```
SET SERVEROUTPUT ON
SET VERIFY OFF

ACCEPT codigo NUMBER PROMPT 'Introduzca el código: ';

DECLARE
BEGIN
    UPDATE estudiante SET promedio = 0
    WHERE codigo = &codigo;
END;
```


VARIABLES en PL/SQL

Uso de las variables

Las variables se usan para:

- Almacenamiento temporal de los datos**
- Manipulación de valores almacenados,**
- Reusabilidad (se pueden utilizar repetidas veces dentro de una aplicación)**

Gestión de Variables en PL/SQL

- Declarar e inicializar las variables dentro de la sección declaradora
- Asignar nuevos valores a las variables dentro de la sección de código
- Pasar valores a los bloques PL/SQL a través de los parámetros
- Ver los resultados a través de variables de salida.

Declaración de Variables PL/SQL

Sintaxis

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

Ejemplos

```
Declare  
  v_fecha          DATE;  
  v_deptno         NUMBER(2) NOT NULL := 10;  
  v_ciudad         VARCHAR2(13) := 'Cali';  
  c_impuesto       CONSTANT NUMBER := 1400;
```

Recomendaciones Declaración Variables

- Seguir las convenciones de nombres
- Inicializar las ctes. y variables designadas como NOT NULL
- Inicializar usando el parámetro de asignación := o la palabra DEFAULT
- Declarar como máximo un identificador por línea

Principales variables escalares

VARCHAR2 (*maximum_length*)

NUMBER [(*precision*, *scale*)]

DATE

CHAR [(*maximum_length*)]

LONG

LONG RAW

BOOLEAN (true,false or NULL)

PLS_INTEGER

Inicialización de variables

USO:

- **:=** Operador de asignación
- **DEFAULT**
- **NOT NULL**

Declaración de variables escalares

Ejemplos

```
DECLARE
  v_nombre          VARCHAR2(9) ;
  v_coont           BINARY_INTEGER := 0 ;
  v_total_sal       NUMBER(9,2) := 0 ;
  v_fecha           DATE := SYSDATE + 7 ;
  c_imp             CONSTANT NUMBER(3,2) := 8.25 ;
  v_valido          BOOLEAN NOT NULL := TRUE ;
  ...
```

El atributo %TYPE

Declarar una variable basada en:

- **Otras variables previamente declaradas**
- **La definición de una columna de la bbdd**

Preceder %TYPE por:

- **La tabla y la columna de la bbdd**
- **El nombre de la variable definida con anterioridad**

Declaración de variables con el Atributo %TYPE

Ejemplos:

```
...  
  v_nombre          empleado.nombre%TYPE;  
  v_balance         NUMBER(7,2);  
  v_min_balance     v_balance%TYPE := 10;  
...
```

Sentencias SQL en PL/SQL

- **PL/SQL soporta:**

Sentencias SQL, para extraer información o aplicar cambios a la bdd

- **PL/SQL no soporta:**

- El lenguaje de definición de datos (DDL), como CREATE TABLE, ALTER TABLE o DROP TABLE

- Lenguaje de control de datos (DCL), como GRANT y REVOKE

Sentencias SELECT de PL/SQL

- **INTO** : Extraer una fila de datos de la bbdd utilizando el comando SELECT y almacenar los datos en una o más variables. Sólo puede ser devuelta una fila

Sintaxis:

```
SELECT    columnas
INTO      {variable1[, variable2]...
            | nombre_registro}
FROM      tabla
WHERE     condicion;
```

Sentencias SELECT de PL/SQL

Ejemplo:

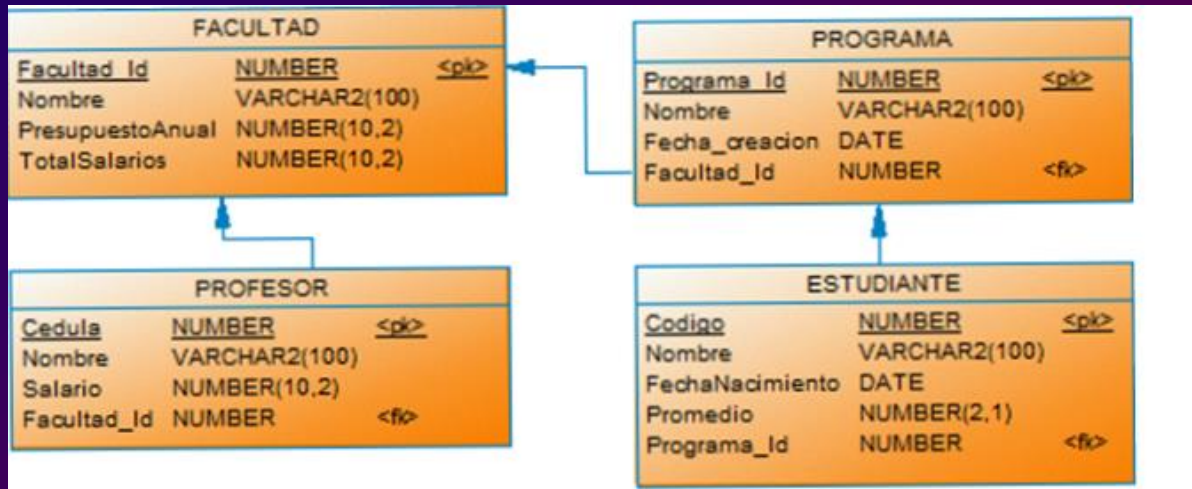
```
DECLARE
    v_deptno          NUMBER(4) ;
    v_ciudad_id       NUMBER(4) ;
BEGIN
    SELECT      departamento_id, ciudad_id
    INTO        v_deptno, v_ciudad_id
    FROM        departamento
    WHERE       department_name = 'Ventas';
    ...
END;
/
```

Sentencias SELECT de PL/SQL

Ejemplo:

```
DECLARE
    v_fecha_ing    empleado.fecha_ing%TYPE;
    v_salario      empleado.salario%TYPE;
BEGIN
    SELECT    fecha_ing, salario
    INTO      v_fecha_ing, v_salario
    FROM      empleado
    WHERE     empleado_id = 100;
    ...
END;
/
```


Ejercicio



Cree un bloque anónimo que imprima en pantalla el número de estudiantes con promedio mayor a 3.

Solución

```
SET SERVEROUTPUT ON

DECLARE
  v_numest number;

BEGIN
  select count(*) into v_numest
  from ESTUDIANTE
  where promedio > 3;

  DBMS_OUTPUT.PUT_LINE(v_numest);
END;
```

Inserción de Datos

Añadir nuevos registros a tabla de bbdd

Ejemplo:

```
BEGIN
  INSERT INTO empleado
    (empleado_id, nombre, apellido, email,
     fecha_ing, trabajo_id, salario)
  VALUES
    (empleado_seq.NEXTVAL, 'Pedro', 'Reyes', 'preyes',
     sysdate, 1, 4000);
END;
/
```

Actualización de datos

Ejemplo:

```
DECLARE
  v_sal_incremento empleado.salario%TYPE := 800;
BEGIN
  UPDATE      empleado
  SET         salario = salario + v_sal_incremento
  WHERE       trabajo_id = 1;
END;
/
```

Supresión de datos

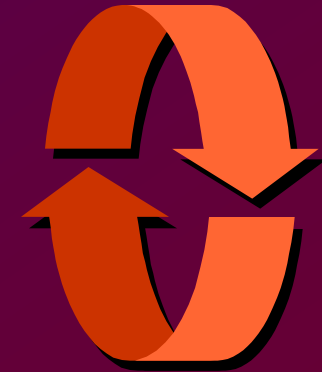
Ejemplo:

```
DECLARE
  v_deptno    empleado.departamento_id%TYPE := 10;
BEGIN
  DELETE FROM    empleado
  WHERE          departamento_id = v_deptno;
END;
/
```

Creación de Estructuras de Control

Control del Flujo de Ejecución PL/SQL

- Se puede modificar el flujo lógico de sentencias utilizando sentencias IF condicionales y estructuras de control de bucles
- Sentencias IF condicionales:
 - IF-THEN-END IF
 - IF-THEN-ELSE-END IF
 - IF-THEN-ELSIF-END IF



Sentencias IF

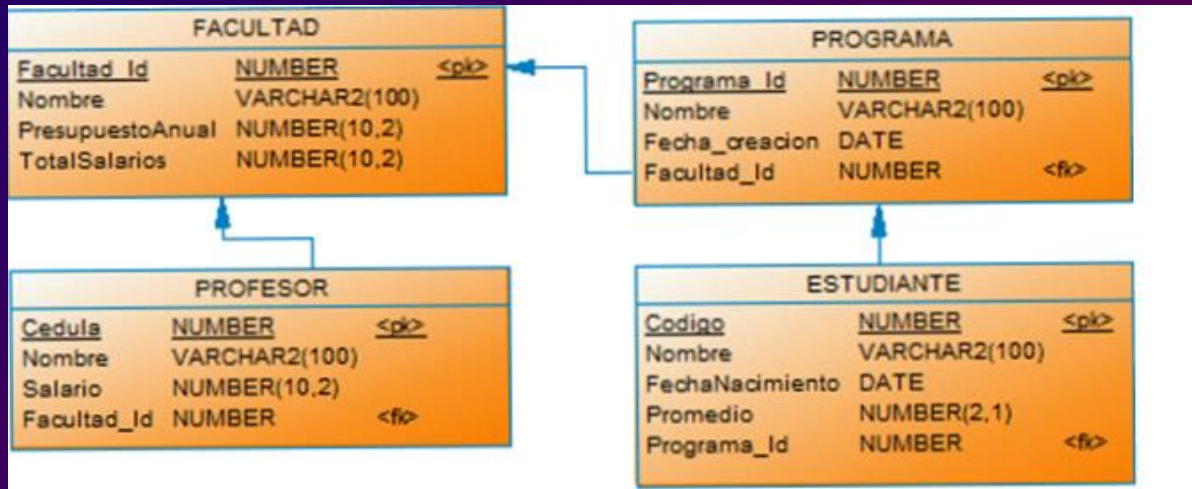
Sintaxis

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

Asignar el ID de director 22 si el apellido del empleado es PEREZ

```
IF v_apellido = 'PEREZ' THEN
    v_dir := 22;
END IF;
```

Ejercicio



Cree un bloque anónimo que imprima en pantalla el nombre del profesor con cédula 1 y su salario. Si el salario es menor a 1'000.000 se imprime el mensaje 'Bajo salario'.

Solución

```
SET SERVEROUTPUT ON

DECLARE
  v_sal PROFESOR.SALARIO%TYPE;
  v_nom PROFESOR.NOMBRE%TYPE;

BEGIN
  select NOMBRE, SALARIO into
         v_nom, v_sal
  from PROFESOR
  WHERE CEDULA = 1;

  DBMS_OUTPUT.PUT_LINE('Nombre: ' || v_nom);
  DBMS_OUTPUT.PUT_LINE('Salario: ' || v_sal);

  IF v_sal < 1000000 THEN
    DBMS_OUTPUT.PUT_LINE('Bajo salario');

  END IF;

END;
```

Sentencia CASE (I)

- La instrucción CASE es una evolución en el control lógico.
- Se diferencia de las estructuras IF-THEN-ELSE en que se puede utilizar una estructura simple para realizar selecciones lógicas en una lista de valores.
- Puede utilizarse también para establecer el valor de una variable

```
CASE variable  
    WHEN expresión1 then valor1  
    WHEN expresión2 then valor2  
    WHEN expresión3 then valor3  
    WHEN expresión4 then valor4  
    ELSE valor5  
END ;
```

Sentencia CASE (II)

Ejemplo

```
SET SERVEROUTPUT ON

DECLARE
  v_equipo varchar2(100);
  v_ciudad varchar2(50);
BEGIN
  v_ciudad := 'PARIS';
  v_equipo:= CASE v_ciudad
              WHEN 'MADRID' then 'RealMadrid'
              WHEN 'BARCELONA' then 'FCBarcelona'
              WHEN 'PARIS' then 'PSG'
              ELSE 'SIN EQUIPO'
            END;
  DBMS_OUTPUT.PUT_LINE(v_equipo);
END;
```

Sentencia CASE (III)

Cada cláusula WHEN puede tener su propia expresión a evaluar. En este caso, después del CASE no aparece ninguna expresión.

Ejemplo

```
SET SERVEROUTPUT ON
DECLARE
  v_salario NUMBER;
  v_tipo NUMBER;
BEGIN
  v_salario := 100;
  CASE
    WHEN v_salario < 10 THEN v_tipo := 1;
    WHEN v_salario < 50 THEN v_tipo := 2;
    ELSE v_tipo := 3;
  END CASE;
  DBMS_OUTPUT.PUT_LINE(v_tipo);
END;
```

Bucle básico

Syntax

```
LOOP                                -- Inicio
  sentencial;<                      -- sentencias
  . . .                             -- sentencia de salida
  EXIT [WHEN condición];          -- Fin
END LOOP;
```

Condición es una expresión booleana

Bucle Básico

Ejemplos

```
-- Bucle infinito
SET SERVEROUTPUT ON
DECLARE
  v_cont NUMBER :=0;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE(v_cont);
    v_cont := v_cont + 1;
  END LOOP ;
END;
```


```
-- Números hasta el 10
SET SERVEROUTPUT ON
DECLARE
  v_cont NUMBER :=0;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE(v_cont);
    v_cont := v_cont + 1;
    EXIT WHEN v_cont > 10;
  END LOOP ;
END;
```

```
-- Números hasta el 10
SET SERVEROUTPUT ON
DECLARE
  v_cont NUMBER :=0;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE(v_cont);
    v_cont := v_cont + 1;
    IF v_cont > 10 THEN
      EXIT;
    END IF;
  END LOOP ;
END;
```


Bucle WHILE

Sintaxis

```
WHILE condición LOOP  
    sentencia1;  
    sentencia2;  
    . . .  
END LOOP;
```



La condición
se evalúa al
principio de
cada iteración

```
SET SERVEROUTPUT ON  
DECLARE  
v_cont NUMBER :=0;  
BEGIN  
    WHILE v_cont <= 10 LOOP  
        DBMS_OUTPUT.PUT_LINE(v_cont);  
        v_cont := v_cont + 1;  
    END LOOP;  
END;
```

Bucle FOR

Sintaxis

```
FOR contador in [REVERSE]  
    valor_inicial..valor_final LOOP  
    sentencia1;  
    sentencia2;  
    . . .  
END LOOP;
```

- Utilizar un bucle FOR cuando se conoce el número de repeticiones
- No declarar el contador, se declara implícitamente

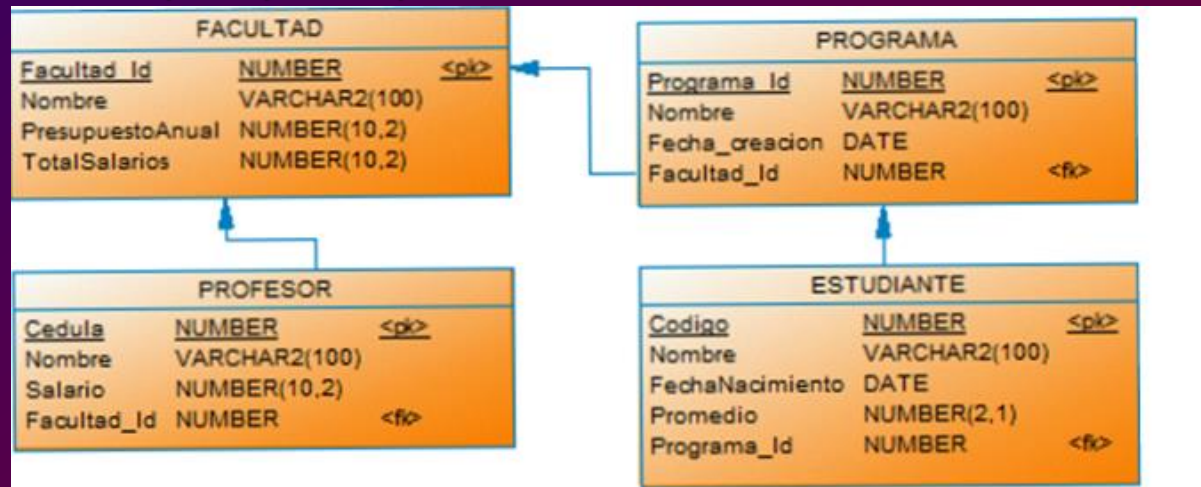
Bucle FOR

Insertar las 10 primeras líneas del pedido del num 101

```
. . .  
    v_ordid      item.ordid%TYPE:=101;  
BEGIN  
. . .  
FOR i IN 1..10 LOOP  
    INSERT INTO item(ordid,itemid)  
    VALUES (v_ordid, i);  
END LOOP;  
END;  
/
```

Ejercicios

- Cree un bloque anónimo que pida al usuario el código de un estudiante e imprima su nombre
- Cree un procedimiento almacenado que pida al usuario el código del estudiante e imprima en pantalla el nombre de la facultad a la que pertenece.
- Cree un procedimiento almacenado que pida al usuario el id de la facultad y si el presupuesto anual es nulo o cero muestre el mensaje “Sin presupuesto”, en caso contrario muestra el presupuesto asignado.
- Cree un procedimiento almacenado que calcule la suma de los salarios de todos los profesores de una facultad (el id de la facultad se recibe como parámetro). Si la suma es mayor que el presupuesto anual de la facultad se muestra el mensaje “Sin presupuesto para salarios” , en caso contrario se muestra “Con presupuesto para salarios”



Bibliografía

**Oracle® Database PL/SQL Language
Reference -11g Release 1 (11.1) - 2009**