

INGENIERIA DE DATOS

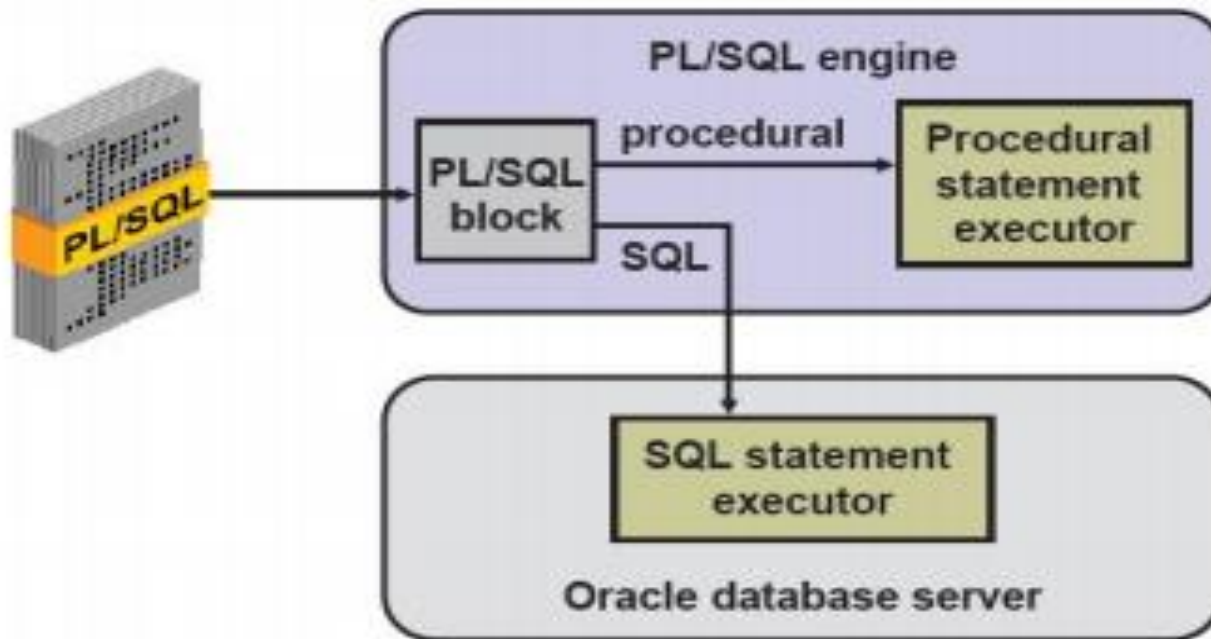
- Programación con PL/SQL
- Procedures y Funciones

PL/SQL

- PLSQL es una extensión de programación a SQL.
- PL/SQL es el lenguaje de programación de 4ta generación para base de datos Oracle
- Todo código PLSQL se compone de código PLSQL+ sentencias SQL.
- Existen problemas cuya solución puede ser más “sencilla” y eficiente mediante un lenguaje procedimental que mediante SQL “puro”
- Ventajas:
 - Permite crear programas modulares.
 - Integración con herramientas de Oracle.
 - Portabilidad.
 - Manejo de errores

Procesamiento de Código

El código PLSQL es ejecutado en un engine motor llamado PLSQL y las secciones que son sentencias SQL son ejecutadas en el SQL Statement Executor (Oracle Database Server).



Procesamiento de Consultas

Se muestra las diferentes secciones que componen los diferentes tipos de código PLSQL.

Anonymous

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

Procedure

```
PROCEDURE name
IS
BEGIN
  --statements

[EXCEPTION]

END;
```

Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

Los procedimientos y funciones llevan a cabo tareas específicas, y su mayor diferencia radica en que las funciones devuelven un valor (return)

Bloque Anonymous PL/SQL

Un bloque PL/SQL es una pieza de código dividida en tres secciones:

DECLARE

Sección de declaración

BEGIN

Sección ejecutable

EXCEPTION

Sección de manejo de excepciones

END;

- Las secciones de **manejo de excepciones** y de **declaración** son opcionales.
- Los bloques pueden contener otros bloques (**sub-bloques**)
- Los comentarios van entre **/* */**. Si no ocupan más de una línea, se pueden escribir después de **--** (dos guiones).

Alcance

```
DECLARE
```

```
  a NUMBER(2) := 10;
```

```
BEGIN
```

```
  DBMS_OUTPUT.PUT_LINE('Valor de a externa ' || a);
```

```
    DECLARE
```

```
    a NUMBER(3) := 20;
```

```
    BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE('Valor de a interna ' || a);
```

```
    END;
```

```
  DBMS_OUTPUT.PUT_LINE('Valor de a ' || a);
```

```
END;
```

/→ Para ejecutar en SQL*Plus

Imprime:

Valor de a externa 10

Valor de a interna 20

Valor de a 10

Operador de concatenación ||

Sub-bloque

Nota: Para ver los resultados de la impresión en **SQL*Plus** se debe ejecutar:

SQL> SET SERVEROUTPUT ON



Ejemplo de bloque simple

DECLARE

nom emp.nom%TYPE;

sue emp.sueldo%TYPE;

cuantos NUMBER(8);

BEGIN

SELECT nom, sueldo **INTO** **nom**, **sue**

FROM emp WHERE cod = 4329;

DBMS_OUTPUT.PUT_LINE('El empleado ' || **nom** || '
tiene sueldo ' || **sue**);

SELECT COUNT(*) **INTO** cuantos

FROM emp;

DBMS_OUTPUT.PUT_LINE('Total empleados ' || cuantos);

END;

/



Luego se
verá como
enviar
parámetros

Declaraciones de variables

La sección de declaración de variables se define en la sección DECLARE ejemplo:

DECLARE

mivar <tipo_dato>

Los tipos de datos en Oracle Database 11g en PLSQL:

CHAR, VARCHAR, NUMBER, BINARY_INTEGER:

Data Type	Description
CHAR [(<i>maximum_length</i>)]	Base type for fixed-length character data up to 32,767 bytes. If you do not specify a maximum length, the default length is set to 1.
VARCHAR2 (<i>maximum_length</i>)	Base type for variable-length character data up to 32,767 bytes. There is no default size for VARCHAR2 variables and constants.
NUMBER [(<i>precision</i> , <i>scale</i>)]	Number having precision <i>p</i> and scale <i>s</i> . The precision <i>p</i> can range from 1 through 38. The scale <i>s</i> can range from -84 through 127.
BINARY_INTEGER	Base type for integers between -2,147,483,647 and 2,147,483,647

El tipo de dato boolean permite los valores null, true y false
permite operaciones lógicas

Consideraciones en PL/SQL

- En PL/SQL se puede usar directamente el sublenguaje de manipulación de datos **DML** de SQL, es decir, **INSERT**, **DELETE**, **UPDATE**, **SELECT** (el **SELECT** requiere usar **INTO** o estar asociado con un cursor, ver luego).
- Para usar sentencias **DDL** en PL/SQL, es decir, **CREATE**, **DROP**, **ALTER** se puede hacer así:

```
BEGIN  
EXECUTE IMMEDIATE 'CREATE TABLE t(ced NUMBER(8))';  
END;  
/
```



Funciones dentro de un programa

- En PL/SQL, las funciones numéricas (SQRT, ROUND, POWER, etc.), de caracteres (LENGTH, UPPER, INITCAP, SUBSTR, etc.) , de fechas (ADD_MONTHS, MONTHS_BETWEEN, etc.); **se pueden usar por fuera de una sentencia SQL** pero las **funciones de grupo** (COUNT, SUM, AVG, MAX, etc.) **solo se pueden usar dentro de una sentencia SQL.**

→ Revisando estructura de Programas

fi queda de tipo **fecha_ing** de **emp**

DECLARE

fi emp.fecha_ing%TYPE;

nom VARCHAR2(20) := INITCAP('adam duritz');

BEGIN

fi := ADD_MONTHS(SYSDATE, -14);

INSERT INTO emp

VALUES (4329,

SUBSTR(**nom**, 1, 8),

fi,

10000);

END;

**Las vbles. se pueden
inicializar en el DECLARE
o en el BEGIN**

**Acá también se
pueden colocar los
valores directamente
y prescindir de las
variables.**

→ Operaciones dentro del programa

DECLARE

nom emp.nom%TYPE;

sue emp.sueldo%TYPE;

cuantos NUMBER(8);

BEGIN

SELECT nom, sueldo INTO nom, sue

FROM emp WHERE cod = 4329;

DBMS_OUTPUT.PUT_LINE('El empleado ' || nom || '
tiene sueldo ' || sue);

SELECT COUNT(*) INTO cuantos

FROM emp;

DBMS_OUTPUT.PUT_LINE('Total empleados ' || cuantos);

END;

/



Luego se
verá como
enviar
parámetros



Control de Datos

La sentencia IF tiene la sintaxis:

```
IF condición THEN
    secuencia de instrucciones
[ELSIF condición THEN
    secuencia de instrucciones]
--Los ELSIF se pueden repetir
[ELSE
    secuencia de instrucciones]
END IF;
```



Ciclos

a) Ciclo simple sin límite: **LOOP**

LOOP

secuencia de instrucciones

END LOOP;

Para salir del ciclo se usa:

EXIT [WHEN condición];



Ciclos

EJEMPLO:

```
DECLARE
  cont NUMBER(4) := 0;
BEGIN
  DELETE plana;
  LOOP
    INSERT INTO plana VALUES(cont, 'Debo aprender
    PL/SQL' || CEIL(DBMS_RANDOM.VALUE(1, 100000)));
    cont := cont + 1;
    EXIT WHEN cont = 1000;
  END LOOP;
END;
```

/



Loop o Ciclos

b) Ciclo para: FOR

Permite repetir una secuencia de instrucciones un número fijo de veces. Su sintaxis es:


```
FOR índice IN [REVERSE] entero .. entero LOOP
    secuencia de instrucciones
END LOOP;
```

Notas:

- El incremento del FOR siempre es 1.
- Aunque el ciclo se haga “en reversa” los límites siempre se colocan de menor a mayor.

Ejemplo de Programa

```
BEGIN
DELETE plana;
  FOR i IN REVERSE 1..500 LOOP
    INSERT INTO plana
      VALUES (i, 'El operador de asignacion en
        PL/SQL es :=' );
  END LOOP;
END;
/
```

The logo of the University of Lima is visible in the background. It is a circular seal with the text 'UNIVERSIDAD DE LIMA' around the top and 'SCIENTIA ET PRAXIS' around the bottom. In the center is a shield with a sunburst and two birds flanking it.

Procedures/ Procedimientos

Sintaxis Procedimientos

CREATE [OR REPLACE] **PROCEDURE** [esquema].**nombre-procedimiento** (nombre-parámetro {IN | OUT | IN OUT} tipo de dato, ..) {IS | AS}

Declaración de variables;

Declaración de constantes;

Declaración de cursores;

BEGIN

Cuerpo del subprograma PL/SQL;

EXCEPTION

Bloque de excepciones PL/SQL;

END;

Functions/ Funciones

Sintaxis Funciones

CREATE [OR REPLACE] **FUNCTION** [esquema].**nombre-función**

(nombre-parámetro {IN | OUT | IN OUT} tipo-de-dato, ...)

RETURN tipo-de-dato {IS | AS}

Declaración de variables;

Declaración de constantes;

Declaración de cursores;

BEGIN

Cuerpo del subprograma PL/SQL;

EXCEPTION

Bloque de excepciones PL/SQL;

END;

Sintaxis de los parámetros

Descripción de la sintaxis:

- **Nombre-parámetro:** es el nombre que queramos dar al parámetro. Podemos utilizar múltiples parámetros. En caso de no necesitarlos, podemos omitir los paréntesis.
- **IN:** especifica que el parámetro es de entrada y que por tanto dicho parámetro tiene que tener un valor en el momento de llamar a la función o procedimiento. Si no se especifica nada, los parámetros son por defecto de tipo entrada.
- **OUT:** especifica que se trata de un parámetro de salida. Son parámetros cuyo valor es devuelto después de la ejecución el procedimiento al bloque PL/SQL que lo llamó. Las funciones PLSQL no admiten parámetros de salida.
- **IN OUT:** Son parámetros de entrada y salida a la vez.
- **Tipo-de-dato:** Indica el tipo de dato PLSQL que corresponde al parámetro (NUMBER, VARCHAR2, etc).

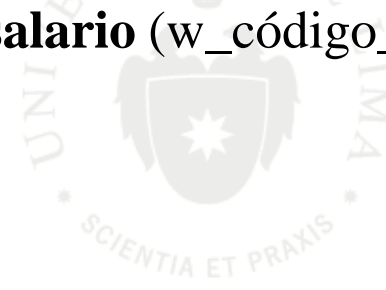
Ejemplo de creación de un procedimiento

```
CREATE OR REPLACE PROCEDURE contratar_empleado  
  (w_codigo_emp IN emp.codigo_emp%TYPE,  
   w_depart IN emp.cod_depart%TYPE,  
   w_fecha_alta IN emp.fecha_alta%TYPE) IS  
BEGIN  
  INSERT INTO emp(código_emp, fecha_alta, cod_depart)  
  VALUES (w_código_emp, w_fecha_alta, w_depart);  
END contratar_empleado;
```

Utilizamos los mismos tipos de dato de la tabla “emp”, es decir: nombreParametro IN nombreTabla.nombreColumna%TYPE. Sería equivalente a poner: w_codigo_emp number, w_depart varchar..

Ejemplo de creación de una función

```
CREATE OR REPLACE FUNCTION obtener_salario (w_código_emp IN  
    emp.código_emp%TYPE)  
RETURN NUMBER  
IS  
    w_salario emp.salario_emp%TYPE;  
BEGIN  
    SELECT salario_emp INTO w_salario  
    FROM emp  
    WHERE código _emp = w_código_emp;  
  
    RETURN w_salario;  
END obtener_salario;
```

The logo of the University of Lima is visible in the background. It features a circular emblem with a shield in the center containing a star. The text "UNIVERSIDAD DE LIMA" is written around the top half of the circle, and "SCIENTIA ET PRAXIS" is written around the bottom half.

Llamadas a procedimientos

- Desde otro procedimiento, función y triggers

```
CREATE PROCEDURE proceso ... IS ...  
BEGIN
```

```
    /* llamada al procedimiento contratar_empleado */  
    contratar_empleado (2645, 'Contabilidad', '19/12/1999');  
END;
```

- En herramientas de desarrollo de aplicaciones de Oracle:
SQL*Plus, SQL*Db, SQL*Forms, SQL*Menu,
SQL*ReportWriter, etc.

```
EXECUTE contratar_empleado (2645, 'Contabilidad',  
'19/12/1999');
```

Llamadas a funciones

Podemos llamar a funciones de distintas formas, por ejemplo:

1. Desde otro procedimiento, función o trigger:

```
CREATE PROCEDURE proceso ... IS ...  
BEGIN ...  
    /* llamada a la función obtener_salario */  
    w_sal :=obtener_salario (w_código);  
END;
```


Llamadas a funciones (cont.)

2. Desde un bloque anónimo

BEGIN

```
DBMS_OUTPUT.PUT_LINE('Salario cod_emp 1 '||  
    obtener_salario(1));
```

END;

3. Desde una instrucción SQL

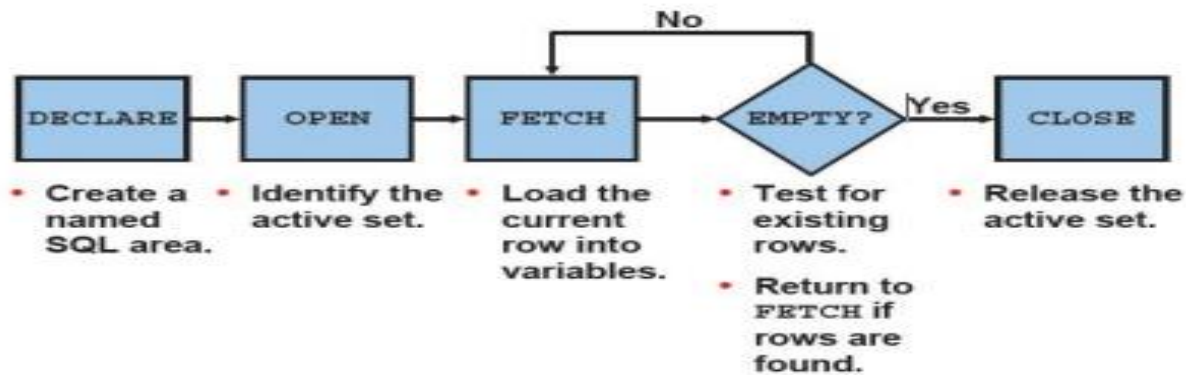
```
SELECT cod_emp, nom_emp, obtener_salario(cod_emp)  
FROM emp;
```

Manejo de Cursores

- Cada sentencia SQL que es ejecutada en la base de datos siempre tiene asociado un cursor.
- Los cursores son definidos en el PGA (Memoria privada por cada server process).
- Los cursores pueden ser definidos por Oracle de forma explícita o implícita.
- Implícita significa que Oracle Database es responsable de crear el cursor, abrirlo recorrerlo y cerrarlo. De forma explícita nosotros somos responsables de las actividades mencionadas.

Control de Cursores

El control de un cursor se define en el siguiente gráfico:



Sintaxis:

DECLARE

CURSOR MICURSOR IS <QUERY>;

BEGIN

OPEN MICURSOR;

FETCH MICURSOR INTO VARIABLES_PLSQL

CLOSE MICURSOR;

END;

/

Control de Cursores

Ejemplo uso de cursor

DECLARE

```
CURSOR cursor_1 IS  
    SELECT nombre, número, salario  
    FROM emp  
    ORDER BY salario;
```

```
w_nombre emp.nombre%TYPE;
```

```
w_número emp.número%TYPE;
```

```
w_salario emp.salario%TYPE;
```

BEGIN

```
    OPEN cursor_1;
```

```
    FETCH cursor_1 INTO w_nombre, w_número, w_salario;
```

```
    ... CLOSE cursor_1;
```

END;



Control de Cursores

```
SET SERVEROUTPUT ON ;
```

```
DECLARE
```

```
    CURSOR MICURSOR IS  
    SELECT FIRST_NAME, LAST_NAME  
    FROM HR.EMPLOYEES;
```

```
BEGIN
```

```
    FOR C IN MICURSOR LOOP
```

```
        DBMS_OUTPUT.PUT_LINE('NOMBRE: '||C.FIRST_NAME||',  
        APELLIDO: '||C.LAST_NAME);
```

```
    END LOOP;
```

```
END;
```

```
/
```



Control de Cursores

Atributos del cursor

Cada cursor definido tiene cuatro atributos a los que se puede acceder para conocer el estado del cursor.

- **%FOUND:** Devuelve true si el último FETCH evaluado devuelve la siguiente fila.
- **%NOTFOUND:** Devuelve true si el último FETCH evaluado no devuelve ninguna fila.
- **%ROWCOUNT:** Contador inicialmente a cero, que se incrementa en uno tras el FETCH de cada fila.
- **%ISOPEN:** Devuelve true si el cursor especificado está abierto.

Manejo de errores

Es la tercera sección, en donde se controlan las acciones si se produjera algún error.

EXCEPTION

WHEN E1 **THEN** --errores definidos dentro del sistema o definidos por el programador.

DBMS_OUTPUT.PUT_LINE('EL QUERY DEVUELVE
MAS DE UNA FILA');

END;

Manejo de errores

```
DECLARE
  nomina NUMBER;
  muy_alta EXCEPTION;
BEGIN
  SELECT SUM(sal) INTO nomina
  FROM emp;
  IF nomina > 100 THEN
    RAISE muy_alta;
  END IF;
  DBMS_OUTPUT.PUT_LINE('El valor es: ' || nomina);
EXCEPTION
  WHEN muy_alta THEN
    DBMS_OUTPUT.PUT_LINE('Nomina muy alta');
END;
/
```



Manejo de errores

Hay dos tipos de excepciones:

1. Excepciones predefinidas por Oracle

- Son un conjunto de errores definidos en Oracle. No hay que declararlas.
- Son disparadas **automáticamente** por PL/SQL como respuesta a un error

2. Excepciones definidas por el usuario

- Se deben declarar
- Se deben disparar **explícitamente**

Manejo de errores

```
CREATE TABLE apuesta(cod NUMBER(8) PRIMARY KEY,  
                      cant NUMBER(8) NOT NULL);
```

```
DECLARE
```

```
  cod_apt_a apuesta.cod%TYPE;
```

```
  nro_ale apuesta.cant%TYPE;
```

```
BEGIN
```

```
  FOR i IN 1..20 LOOP
```

```
    BEGIN
```

```
      cod_apt_a := ABS(MOD(DBMS_RANDOM.RANDOM, 10));
```

```
      nro_ale := ABS(MOD(DBMS_RANDOM.RANDOM, 10));
```

```
      INSERT INTO apuesta VALUES(cod_apt_a, nro_ale);
```

```
      EXCEPTION
```

```
      WHEN DUP_VAL_ON_INDEX THEN
```

```
        UPDATE apuesta SET cant = cant + nro_ale
```

```
        WHERE cod = cod_apt_a;
```

```
    END;
```

```
  END LOOP;
```

```
END;
```

```
/
```

S
u
b
b
l
o
q
u
e

Manejo de errores

Algunas de las excepciones predefinidas:

- `TOO_MANY_ROWS`
- `INVALID_CURSOR`
(Ej: Cerrar un cursor que ya estaba cerrado)
- `NO_DATA_FOUND`
- `CURSOR_ALREADY_OPEN`
(Ej: Abrir un cursor que ya estaba abierto)
- `INVALID_NUMBER`
(Ej: Fallo de conversión, 'k3b' no es un número)
- `VALUE_ERROR`
(Ej: Error de truncamiento)
- `ZERO_DIVIDE`
- `DUP_VAL_ON_INDEX`

Manejo de errores

```
RAISE exception_name
```

```
IF v_salary > v_max then
```

```
    RAISE salary_too_high;
```

```
End if
```

Jump to the exception section user defined as salary_too_high

Raise_application_error(error_number,error_message)

Error_number is a negative integer in the range -20000 to -20999

Error_message is a character string up to 512 bytes

Manejo de errores

```
EXCEPTION
    WHEN salary_too_high
        then "PL/SQL statements here";
    WHEN another_error OR yet_another
        then "PL/SQL statements here";
    WHEN OTHERS                                -- Oracle
defined
        then "PL/SQL statements here";
END;
```

Others corresponde a la ultima opción de error definida

Manejo de errores

--En este ejemplo 40 no tiene personal

DECLARE

v_salary personnel.salary%type;

BEGIN

select salary into v_salary from personnel where div=40;

EXCEPTION

when too_many_rows -- this is the pre-defined exception

then **raise_application_error (-20001, ' Did not expect so many');**

END;

/

declare

*

ERROR at line 1:

ORA-01403: no data found - occurs as div 40
has no staff!

ORA-06512: at line 4

