

PL/SQL

Francisco Moreno

Universidad Nacional

Introducción a PL/SQL

¿Por qué PL/SQL?

- A pesar de que SQL tiene mecanismos de control condicional (cláusula `CASE WHEN*`) e iterativos (*implícitos*) en ocasiones se requiere:
 - Manipular y controlar los datos de una manera secuencial
 - Mejorar el desempeño de las aplicaciones
- Existen problemas cuya solución puede ser más “sencilla” y eficiente mediante un lenguaje procedimental que mediante SQL “puro”

* PL/SQL también la tiene. Más adelante se ve un ejemplo.

Introducción al PL/SQL

- Incorporación de PSM* a SQL (1992) → Incluye estructuras de **secuencia**, **decisión**, **iteración**, creación de procedimientos, funciones, etc.
- La versión PSM de Oracle se llama PL/SQL (*Procedural Language/SQL*).
En SQL Server se llama Transact-SQL (T-SQL).
- En PL/SQL se pueden crear **procedimientos** con o sin nombre (anónimos), **funciones**, **disparadores** (*triggers*) y bibliotecas de funciones y procedimientos llamadas **paquetes**.

* *Persistent Stored Modules*

Bloques 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*) → ver luego
- Los comentarios van entre */* */*. Si no ocupan más de una línea, se pueden escribir después de *--* (dos guiones).

Variables y constantes

- Tipos de datos* en PL/SQL: NUMBER, CHAR, VARCHAR/VARCHAR2, DATE, BOOLEAN, entre otros.
- La sintaxis para declarar variables o constantes es:

nomvar [CONSTANT] TIPO [NOT NULL] [:= expresión];

Los
corchetes
indican las
partes
opcionales

- En los nombres de las variables no se diferencian mayúsculas y minúsculas.

* NUMBER, CHAR, VARCHAR/VARCHAR2 tienen precisión.

- Se pueden declarar variables refiriéndose al tipo de datos de otros elementos tales como variables, columnas y tablas, ver ejemplos más adelante.
- El operador de asignación es `:=` y el de igualdad es `=`.

Alcance

El alcance o visibilidad de las variables sigue estas reglas:

1. Una variable es visible en el bloque en el cual se declara y en todos sus sub-bloques, a menos que se aplique la regla 2.
2. Si se declara una variable en un sub-bloque con el mismo nombre que una variable del bloque contenedor (externo), la variable del sub-bloque es la que tiene prioridad en el sub-bloque*.

* Es posible acceder en el sub-bloque a la variable del bloque contenedor (externo) mediante etiquetas (luego se ejemplifican), **pero lo más sencillo es usar nombres diferentes para las variables.**

Alcance

Operador de concatenación ||

```
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;
```

Sub-
bloque

```
  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

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

SQL> SET SERVEROUTPUT ON

- 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 (nro
NUMBER(3) PRIMARY KEY)';

END;

/

- La **sentencia DDL NO** lleva punto y coma dentro de las comillas simples.
- Lo que sigue a IMMEDIATE puede ser una variable de caracteres → Luego se verán más ejemplos

- Nota: Usar EXECUTE IMMEDIATE solo cuando sea indispensable, lo siguiente es innecesario, aunque funciona:

```
BEGIN  
EXECUTE IMMEDIATE ' INSERT INTO t VALUES (97) ' ;  
END;  
/
```

- Es más sencillo y eficiente así:

```
BEGIN  
INSERT INTO t VALUES (15) ;  
END;  
/
```

- 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.

Ejemplo

```
DROP TABLE emp;  
CREATE TABLE emp(  
    cod NUMBER(8) PRIMARY KEY,  
    nom VARCHAR2(10) NOT NULL,  
    fecha_ing DATE,  
    sueldo NUMBER(8) CHECK(sueldo > 0)  
);
```

`fi` queda de tipo `fecha_ing` de `emp`

DECLARE

`fi` emp.fecha_ing%TYPE;

`nom` VARCHAR2(20) := INITCAP('carmen electra');

BEGIN

`fi` := ADD_MONTHS(SYSDATE, -14);

INSERT INTO emp

VALUES (4329,

SUBSTR(`nom`, 1, 10),

`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.**

Sobre las consultas SQL en PL/SQL:

- Se debe proporcionar un “lugar” para guardar los datos devueltos por una consulta (**SELECT**)
- Esto se puede lograr mediante la cláusula **SELECT ... INTO**.
- Sin embargo, un **SELECT ... INTO** debe retornar **una y solo una fila**:
 - Si la consulta no recupera filas o recupera varias filas, ocurre un error (**excepción**, se verán luego).
 - Los **cursores** (**se ven luego**) sirven para consultas que recuperan 0, 1 o más filas.

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 Flujo

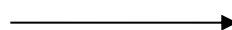
- Las comparaciones lógicas son la base del control condicional en PL/SQL.

Los resultados de las comparaciones son verdadero (TRUE), falso (FALSE) o nulo (NULL).

- Cualquier “cosa” comparada con NULL retorna NULL (**desconocido**).
- Los operadores lógicos son: =, >, <, >=, <=, !=, <>

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;
```



Ingresa
acá si la
condición
fue TRUE

Comparación con nulo: ¿Qué imprime el siguiente programa?:

```
DECLARE
  a NUMBER := NULL;
BEGIN
  IF a = a THEN
    DBMS_OUTPUT.PUT_LINE(' 0 sea que NULL = NULL' );
  ELSIF a <> a THEN
    DBMS_OUTPUT.PUT_LINE(' 0 sea que NULL <> NULL' );
  ELSE
    DBMS_OUTPUT.PUT_LINE(' Indefinido, NULL no es ni = ni
    <> a NULL' );
  END IF;
END;
/
```

Lo anterior también se puede escribir con CASE así:

```
DECLARE
```

```
  a NUMBER := NULL;
```

```
BEGIN
```

```
CASE
```

```
WHEN a = a THEN
```

```
  DBMS_OUTPUT.PUT_LINE(' 0 sea que NULL = NULL');
```

```
WHEN a <> a THEN
```

```
  DBMS_OUTPUT.PUT_LINE(' 0 sea que NULL <> NULL');
```

```
ELSE
```

```
  DBMS_OUTPUT.PUT_LINE(' Indefinido, NULL no es ni =  
ni <> a NULL');
```

```
END CASE;
```

```
END;
```

```
/
```

Ciclos o iteraciones

a) Ciclo simple sin límite: LOOP

LOOP

secuencia de instrucciones

END LOOP;

Para salir del ciclo se usa:

EXIT [WHEN condición];

Ejemplo.

```
DROP TABLE plana;
```

```
CREATE TABLE plana(nro NUMBER(4) PRIMARY KEY,  
dato VARCHAR2(80));
```

```
DECLARE
```

```
    cont NUMBER(4) := 0;
```

```
BEGIN
```

```
DELETE plana;
```

```
LOOP
```

```
    INSERT INTO plana VALUES(cont, 'No traer a Ariana  
Grande ' || CEIL(DBMS_RANDOM.VALUE(1, 5000)));
```


```
    cont := cont + 1;
```

```
EXIT WHEN cont = 1000;
```

```
END LOOP;
```

```
END;
```

```
/
```



CEIL redondea hacia al
próximo entero

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. Veamos un ejemplo:

Ejemplo:

```
BEGIN
DELETE plana;
  FOR cont IN REVERSE 1..500 LOOP
    INSERT INTO plana
      VALUES (cont, 'Traer a Carmen Electra');
  END LOOP;
END;
/
```


c) Ciclo mientras que: WHILE

WHILE repetirá una secuencia de instrucciones hasta que la condición controladora del ciclo deje de ser cierta. Su sintaxis es:

```
WHILE condición LOOP
    secuencia de instrucciones
END LOOP;
```

Ejemplo:

```
DECLARE
  cont NUMBER(3) := 500;
BEGIN
  DELETE PLANA;
  WHILE cont > 0 LOOP
    INSERT INTO plana VALUES
      (cont, DBMS_RANDOM.STRING('u', 60) || cont);
    cont := cont - 1;
  END LOOP;
END;
```

/

u: mayúsculas
l: minúsculas
a: combinación de
mayúsculas y
minúsculas
x: alfanuméricos

Tamaño