

# Apuntes PL/SQL

## Primeros Pasos

### Bloques Anónimos

Los bloques anónimos no se guardan en base de datos, por tanto toca guardarlos en un fichero \*.sql

```
BEGIN
    NULL; -- El comando NULL no hace nada
END;
/ -- El slash final es opcional pero normalmente se pone para indicar el
fin del bloque PL/SQL
```

### Visualizar Salida en Consola

```
SET SERVEROUTPUT ON
BEGIN
    DBMS_OUTPUT.PUT_LINE(100); -- 100
    DBMS_OUTPUT.PUT_LINE('HELLO' || ' WORLD!'); -- HELLO WORLD!
END;
/
```

### Variables

Consideraciones:

- Pueden ser de distintos tipos
- Pueden ser utilizadas en comandos SQL
- Para nombrar una variable:
  - Debe comenzar por una letra
  - Se pueden incluir número o letras
  - Puede tener algunos caracteres especiales: \$ , \_ , etc
  - Longitud de máximo 30 caracteres
  - No se pueden usar palabras reservadas

Declaración:

- Se declaran e inicializan en la sección `DECLARE` del bloque PL/SQL
- Se pueden pasar como argumento de otros bloques PL/SQL
- Pueden almacenar valores retornados por otros programas PL/SQL

```

DECLARE
    SALARY NUMBER(4) := 1000;
    NAME VARCHAR2(100) := 'John Doe';
BEGIN
    SALARY := SALARY*10;
    IF SALARY > 10000 THEN
        ...
        ...
        ...
END;
/

```

## Contantes y NULL

Para el caso de las constantes es necesario siempre inicializarlas y no se puede modificarlas

```

SET SERVEROUTPUT ON
DECLARE
    X CONSTANT NUMBER := 10;
    Z NUMBER NOT NULL := 20;
BEGIN
    DBMS_OUTPUT.PUT_LINE(X);
    X := 100; -- ERROR
    Z := 30;
    DBMS_OUTPUT.PUT_LINE(Z);
    Z := NULL; -- ERROR
END;
/

```

## Variables Boolean

A pesar de que en Oracle SQL no existe un tipo de dato `BOOLEAN`, en PL/SQL sí existe este tipo de dato, es importante tener en cuenta como la sentencia `IF THEN ELSE` trabaja con las condiciones:

```

SET SERVEROUTPUT ON
DECLARE
    IS_ACTIVE BOOLEAN := NULL;
BEGIN
    DBMS_OUTPUT.PUT_LINE (IS_ACTIVE); -- ERROR
    DBMS_OUTPUT.PUT_LINE (CASE WHEN IS_ACTIVE THEN 'TRUE' ELSE 'NOT TRUE'
END ); -- NOT TRUE
    IS_ACTIVE := TRUE;
    DBMS_OUTPUT.PUT_LINE (CASE WHEN IS_ACTIVE THEN 'TRUE' ELSE 'NOT TRUE'
END); -- TRUE
    IS_ACTIVE := FALSE;
    DBMS_OUTPUT.PUT_LINE (CASE WHEN IS_ACTIVE THEN 'TRUE' ELSE 'NOT TRUE'
END); -- NOT TRUE
END;
/

```

## %TYPE

Con este tipo de dato, para una variable es posible declarar el mismo tipo dato que el de una columna de una tabla

Para la tabla REGIONS:

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

```

SET SERVEROUTPUT ON
DECLARE
    REGION REGIONS.REGION_NAME%TYPE;
BEGIN
    REGION := 'Oceanía';
    DBMS_OUTPUT.PUT_LINE (REGION);
END;
/

```

## Operadores

PL/SQL intentará en lo posible hacer la transformación entre tipos de datos de forma implícita. Operadores más habituales:

- Suma: +
- Resta: -
- Multiplicación: \*
- División: /
- Exponente: \*\*
- Concatenación: ||

También podemos hacer operaciones con tipos de dato como fechas

```
SET SERVEROUTPUT ON
DECLARE
    TODAY DATE := SYSDATE;
    MY_BIRTH_DATE DATE := '08-05-1993';
BEGIN
    DBMS_OUTPUT.PUT_LINE(TODAY);      -- Formato dd/mm/yyyy
    TOMORROW = TODAY + 1;
    DBMS_OUTPUT.PUT_LINE(TOMORROW);
END;
/
```

## Comentarios

```
-- Esto es un comentario de una sola línea
/*
    Esto es un comentario de bloque
*/
```

## Bloques Anidados y Scope de las Variables

```
SET SERVEROUTPUT ON
DECLARE
    X NUMBER := 20;  --GLOBAL
    Z NUMBER := 30;
BEGIN
    DBMS_OUTPUT.PUT_LINE('X: ' || X); -- 20
    DECLARE
        X NUMBER := 10;  --LOCAL
        Y NUMBER := 200;
    BEGIN
        DBMS_OUTPUT.PUT_LINE('X: ' || X); -- 10
        DBMS_OUTPUT.PUT_LINE('Z: ' || Z); -- 30
    END;
    DBMS_OUTPUT.PUT_LINE('Y: ' || Y); -- ERROR
```

```
END;  
/
```

## Funciones SQL en PL/SQL

Las funciones SQL que se ejecutan dentro de PL/SQL tienen las mismas funcionalidades que al ejecutarlas en el motor SQL de Oracle, sin embargo, estas se ejecutan en el motor de PL/SQL, sin embargo, no es posible usar funciones de agrupación, ni la función `DECODE`

```
SET SERVEROUTPUT ON  
DECLARE  
    X VARCHAR2(50);  
    MAYUS VARCHAR2(100);  
    FECHA DATE;  
    Z NUMBER := 109.80;  
BEGIN  
    X := 'Ejemplo';  
    DBMS_OUTPUT.PUT_LINE(SUBSTR(X,1,3)); -- Eje  
    MAYUS := UPPER(X);  
    DBMS_OUTPUT.PUT_LINE(MAYUS);          -- EJEMPLO  
    FECHA := SYSDATE;  
    DBMS_OUTPUT.PUT_LINE(FECHA);          -- 06/12/2022  
    DBMS_OUTPUT.PUT_LINE(FLOOR(Z));       -- 109  
END;  
/
```

## Operadores Lógicos y Relacionales

Operadores relacionales o de comparación:

- Igual a: =
- Distinto de: <>
- Menor que: <
- Mayor que: >
- Menor o igual que: <=
- Mayor o igual que: >=

Operadores lógicos:

- AND
- OR
- NOT

# Estructuras de Control

## Sentencia IF

```
SET SERVEROUTPUT ON
DECLARE
    sales    NUMBER := 25000;
    bonus    NUMBER := 0;
BEGIN
    IF sales > 50000 THEN
        bonus := 1500;
    ELSIF sales > 35000 THEN
        bonus := 500;
    ELSIF sales > 20000 THEN
        bonus := 150;
    ELSE
        bonus := 100;
    END IF;
    DBMS_OUTPUT.PUT_LINE('Sales = '
    || sales
    || ', bonus = '
    || bonus
    || '.');
END;
/
```

## Sentencia CASE

```
SET SERVEROUTPUT ON
DECLARE
    v1 CHAR(1);
BEGIN
    v1 := 'B';
    CASE v1
        WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
        WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good'); -- Se imprime esta
línea
        WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
        WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
        WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
        ELSE DBMS_OUTPUT.PUT_LINE('POOR!!!!');
    END CASE;
END;
/
```

## Searched CASE

```
SET SERVEROUTPUT ON
DECLARE
    BONUS    NUMBER;
BEGIN
    BONUS := 100;
    CASE
        WHEN BONUS > 500 THEN DBMS_OUTPUT.PUT_LINE('EXCELLENT');
        WHEN BONUS <= 500 AND BONUS > 250 THEN DBMS_OUTPUT.PUT_LINE('VERY
GOOD');
        WHEN BONUS <= 250 AND BONUS > 100 THEN DBMS_OUTPUT.PUT_LINE('GOOD');
        ELSE DBMS_OUTPUT.PUT_LINE('POOR!!!!');
    END CASE;
END;
/
```

## Bucle LOOP

La sentencia `EXIT` es equivalente a la sentencia `break` de otros lenguajes de programación

```
SET SERVEROUTPUT ON
DECLARE
    X NUMBER:=1;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE(X);
        X:=X+1;
        /*
        IF X = 11 THEN
            EXIT;
        END IF;
        */
        EXIT WHEN X=11;
    END LOOP;
END;
/
```

## LOOP Anidado

La gestión de etiquetas ayuda a manipular el ciclo de vida de los ciclos de una forma más potente

```
DECLARE
    S PLS_INTEGER := 0;
    I PLS_INTEGER := 0;
    J PLS_INTEGER;
```

```

BEGIN
    <<PARENT>>
    LOOP
        I := I + 1;
        J := 100;
        DBMS_OUTPUT.PUT_LINE('PARENT:' || I);
        <<CHILD>>
        LOOP
            --PRINT CHILD
            DBMS_OUTPUT.PUT_LINE('CHILD:' || J);
            J:=J+1;

            EXIT PARENT WHEN (I> 3);
            EXIT CHILD WHEN (J > 105);
        END LOOP CHILD;
    END LOOP PARENT;
    DBMS_OUTPUT.PUT_LINE('FINISH!!!');
END;
/

```

## Sentencia CONTINUE

```

DECLARE
    X NUMBER := 0;
BEGIN
    LOOP -- CON CONTINUE SALTAMOS AQUI
        DBMS_OUTPUT.PUT_LINE ('LOOP:  X = ' || TO_CHAR(X));
        X := X + 1;
        /*
        IF X < 3 THEN
            CONTINUE;
        END IF;
        */
        CONTINUE WHEN X <3;
        DBMS_OUTPUT.PUT_LINE
        ('DESPUES DE  CONTINUE:  X = ' || TO_CHAR(X));
        EXIT WHEN X = 5;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE (' DESPUES DEL  LOOP:  X = ' || TO_CHAR(X));
END;
/

```

## Bucle FOR

Imprime en consola valores del 5 al 15, incluídos, la variable `i` tiene su *scope* únicamente dentro del ciclo.



```

SET SERVEROUTPUT ON
BEGIN
    FOR i IN 5..15 LOOP    -- PLS_INTEGER
        DBMS_OUTPUT.PUT_LINE(i);
    END LOOP;
END;
/

```

## Sentencia REVERSE

Imprime en consola valores del 15 al 5, incluídos

```

SET SERVEROUTPUT ON
BEGIN
    FOR i IN REVERSE 5..15 LOOP    -- PLS_INTEGER
        DBMS_OUTPUT.PUT_LINE(i);
        EXIT WHEN i=10;    -- Siguen funcionando sentencias EXIT y
CONTINUE
    END LOOP;
END;
/

```

## Bucle WHILE

Imprime en consola valores del 0 al 9

```

DECLARE
    X PLS_INTEGER := 0;
BEGIN
    WHILE X<10 LOOP
        DBMS_OUTPUT.PUT_LINE(X);
        X:=X+1;
        EXIT WHEN X=5;
    END LOOP;
END;
/

```

## Sentencia GOTO

Se recomienda evitar su uso, sirve para ir a una ubicación específica dentro del programa denotada por una etiqueta

# SQL en PL/SQL

Para la tabla EMPLOYEES:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
198	Donald	OConnell	DOCONNEL	650.507.9833	21-JUN-07	SH_CLERK	2600		124	50
199	Douglas	Grant	DGRANT	650.507.9844	13-JAN-08	SH_CLERK	2600		124	50
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-03	AD_ASST	4400		101	10
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-04	MK_MAN	13000		100	20
202	Pat	Fay	PFAY	603.123.6666	17-AUG-05	MK_REP	6000		201	20

## SELECT INTO dentro de PL/SQL

El resultado de un `SELECT`, debe ser una sola fila (ni más de una, ni cero), y debe guardarse en una variable

```
SET SERVEROUTPUT ON
DECLARE
    SALARIO NUMBER;
    NOMBRE   EMPLOYEES.FIRST_NAME%TYPE;
BEGIN
    SELECT --SOLO PUEDE DEVOLVER UNA FILA
           SALARY, FIRST_NAME INTO SALARIO, NOMBRE
    FROM
        EMPLOYEES
    WHERE
        EMPLOYEE_ID = 199;
    DBMS_OUTPUT.PUT_LINE(SALARIO); -- 2600
    DBMS_OUTPUT.PUT_LINE(NOMBRE);  -- Douglas
END;
/
```

## %ROWTYPE

```
SET SERVEROUTPUT ON
DECLARE
    EMPLEADO EMPLOYEES%ROWTYPE;
BEGIN
    SELECT --SOLO PUEDE DEVOLVER UNA FILA
           * INTO EMPLEADO
    FROM
        EMPLOYEES
    WHERE
        EMPLOYEE_ID = 100;
    DBMS_OUTPUT.PUT_LINE(EMPLEADO.SALARY*100);
    DBMS_OUTPUT.PUT_LINE(EMPLEADO.FIRST_NAME);
END;
/
```

## INSERT dentro de PL/SQL

Para la tabla `REGIONS`:

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

```
DECLARE
    ID REGIONS.REGION_ID%TYPE;
    NOMBRE REGIONS.REGION_NAME%TYPE;
BEGIN
    ID := 5;
    NOMBRE := 'Oceanía';
    INSERT INTO REGIONS (REGION_ID, REGION_NAME) VALUES (ID, NOMBRE);
    COMMIT;
END;
/
```

## UPDATE dentro de PL/SQL

```
DECLARE
    ID REGIONS.REGION_ID%TYPE;
    NOMBRE REGIONS.REGION_NAME%TYPE;
BEGIN
    ID := 5;
    NOMBRE := 'Antartida';
    UPDATE REGIONS SET REGION_NAME=NOMBRE WHERE REGION_ID=ID;
    COMMIT;
END;
/
```

## DELETE dentro de PL/SQL

```

DECLARE
    ID REGIONS.REGION_ID%TYPE;
BEGIN
    ID := 5;
    DELETE FROM REGIONS WHERE REGION_ID=ID;
    COMMIT;
END;
/

```

## Excepciones

Tipos de excepciones:

- Predefinidas: Ya vienen asociadas a errores propios de Oracle, propias de Oracle
- No predefinidas: Asociaciones a errores propios de Oracle, definidas por el usuario
- Personalizadas: Definidas por el usuario

### Predefinidas

Las sentencias `SQLCODE` y `SQLERRM` sirven para poder visualizar el código y el mensaje de error de las excepciones de forma más sencilla, ambas son funciones PL/SQL, es decir, no correrán en el motor SQL de Oracle

```

SET SERVEROUTPUT ON
DECLARE
    EMPL EMPLOYEES%ROWTYPE;
BEGIN
    SELECT * INTO EMPL
    FROM EMPLOYEES
    WHERE EMPLOYEE_ID>1;
    DBMS_OUTPUT.PUT_LINE (EMPL.FIRST_NAME);
EXCEPTION
    -- NO_DATA_FOUND      ORA-01403
    -- TOO_MANY_ROWS      ORA-01422
    -- ZERO_DIVIDE
    -- DUP_VAL_ON_INDEX
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('ERROR, EMPLEADO INEXISTENTE');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE ('ERROR, DEMASIADOS EMPLEADO');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE ('ERROR INDEFINIDO');
        DBMS_OUTPUT.PUT_LINE (SQLCODE);
        DBMS_OUTPUT.PUT_LINE (SQLERRM);
END;
/

```

## No Predefinidas

Para definir una excepción asociada a errores existentes en Oracle, es necesario conocer el código de dichos errores, este se puede consultar en: <https://docs.oracle.com/en/database/oracle/oracle-database/21/errmg/index.html>

El siguiente ejemplo busca manejar la excepción generada por un error al tratar de usar funciones de agrupación en una sentencia `SELECT INTO` simple:

```
SET SERVEROUTPUT ON
DECLARE
    MI_EXCEP EXCEPTION;
    PRAGMA EXCEPTION_INIT(MI_EXCEP, -937);
    V1 NUMBER;
    V2 NUMBER;
BEGIN
    SELECT EMPLOYEE_ID, SUM(SALARY) INTO V1, V2 FROM EMPLOYEES;
    DBMS_OUTPUT.PUT_LINE(V1);
EXCEPTION
    WHEN MI_EXCEP THEN
        DBMS_OUTPUT.PUT_LINE('FUNCION DE GRUPO INCORRECTA');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('ERROR INDEFINIDO');
        DBMS_OUTPUT.PUT_LINE(SQLCODE);
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
```

## Personalizadas

```
DECLARE
    REG_MAX EXCEPTION;
    REGN NUMBER;
    REGT VARCHAR2(200);
BEGIN
    REGN:=101;
    REGT:='ASIA';
    IF REGN > 100 THEN
        RAISE REG_MAX;
    ELSE
        INSERT INTO REGIONS VALUES (REGN, REGT);
        COMMIT;
    END IF;
EXCEPTION
    WHEN REG_MAX THEN
        DBMS_OUTPUT.PUT_LINE('LA REGION NO PUEDE SER MAYOR DE 100.');
```

```
END;  
/
```

El *scope* de las excepciones personalizadas y no predefinidas, funciona de la misma forma que las variables en los bloques anidados

```
DECLARE  
    REG_MAX EXCEPTION;  
    REGN NUMBER;  
    REGT VARCHAR2(200);  
BEGIN  
    REGN:=101;  
    REGT:='ASIA';  
    DECLARE  
        REG_MAX EXCEPTION;  
    BEGIN  
        IF REGN > 100 THEN  
            RAISE REG_MAX;  
        ELSE  
            INSERT INTO REGIONS VALUES (REGN,REGT);  
            COMMIT;  
        END IF;  
    EXCEPTION  
        WHEN REG_MAX THEN  
        DBMS_OUTPUT.PUT_LINE('LA REGION NO PUEDE SER MAYOR DE 100.  
BLOQUE HIJO');  
    END;  
EXCEPTION  
    WHEN REG_MAX THEN  
        DBMS_OUTPUT.PUT_LINE('LA REGION NO PUEDE SER MAYOR DE 100.  
BLOQUE PADRE');  
    WHEN OTHERS THEN  
        DBMS_OUTPUT.PUT_LINE('ERROR INDEFINIDO');  
END;  
/
```

## Sentencia `RAISE_APPLICATION_ERROR`

```
DECLARE  
    REGN NUMBER;  
    REGT VARCHAR2(200);  
BEGIN  
    REGN:=101;  
    REGT:='ASIA';  
    IF REGN > 100 THEN  
        -- EL CODIGO DEBE ESTAR ENTRE -20000 Y -20999  
        RAISE_APPLICATION_ERROR(-20001,'LA ID NO PUEDE SER MAYOR DE  
100');
```

```
ELSE
    INSERT INTO REGIONS VALUES (REGN,REGT);
    COMMIT;
END IF;
END;
/
```