EFA
MORATALAZ

1º CFGS Desarrollo de Aplicaciones Web

BASES DE DATOS

ROBERTO SÁNCHEZ CHACÓN

UT5 – PROGRAMACIÓN DE BBDD

EFA
MORATALAZ

1º CFGS Desarrollo de Aplicaciones Web

BASES DE DATOS

UT5 – PROGRAMACIÓN DE BBDD

1. INTRODUCCIÓN AL LENGUAJE DE PROGRAMACIÓN
2. DECLARACIÓN DE VARIABLES
3. INPUT / OUPUT
4. ESTRUCTURAS DE CONTROL DE FLUJO
5. PROCEDIMIENTOS Y FUNCIONES
6. CONTROL DE ERRORES
7. CURSORES
8. DISPARADORES (TRIGGERS)
9. PAQUETES

INTRODUCCIÓN AL LENGUAJE DE PROGRAMACIÓN

- En ORACLE igual que ocurre en otros lenguajes de programación, se permite la declaración de **bloques compuestos**.
- Una declaración compuesta es un bloque que puede contener otros bloques, declaraciones para variables, manejadores de condiciones y cursores; y construcciones de control de flujo tales como bucles y pruebas condicionales.
- El principio y final de un bloque compuesto se define con la sintaxis: **DECLARE … BEGIN … EXCEPTION … END**
- Los bloques compuestos, también se utilizan en el cuerpo de **definición de procedimientos y funciones almacenados y triggers.**
- Estos objetos se definen en términos de código SQL que se almacena en el servidor para su posterior invocación.

- Estructura de bloque anónimo PL/SQL

```
DECLARE
    <sentenciasDeDeclaracion>
BEGIN
     <sentencias>
EXCEPTION
    WHEN <nombreExcepción> THEN
        <sentenciasDeExcepción>
END;
```

DECLARACIÓN DE VARIABLES

- La declaración de variables se realiza en la sección **DECLARE** de un bloque anónimo y tras la declaración de un prototipo de función o procedimiento

- En el momento de la declaración, también se puede asignar un valor haciendo uso el operador **:=**, quedando la sintaxis final como sigue:

nombreVariable **TIPO_DATO :=** <valorAsignado>;

- Una **constante** es una variable la cual no cambia su valor, para poder usar constantes se usa la palabra reservada **CONSTANT**, quedando la sintaxis como sigue:

nombreConstante **CONSTANT TIPO_DATO :=** <valorAsignado>;

- Una forma más correcta de declarar variables es usar los atributos **%TYPE** o **%ROWTYPE**, a la hora de seleccionar un tipo de dato, quedando la sintaxis de la siguiente forma:

nombreVariable **<nombreTabla>.<columna>%TYPE :=** <valorAsignado>;

nombreVariable **<nombreTabla>%ROWTYPE :=** <valorAsignado>;

- Para el caso de una **constate** la nomenclatura quedaría como sigue:

nombreConstante **CONSTANT <nombreTabla>.<columna>%TYPE :=** <valorAsignado>;

- SELECT EN PL/SQL

SELECT campo1, campo2…
INTO Variable1, variable2
FROM tabla
WHERE…

INPUT / OUPUT

- **Lectura de datos:** Para leer datos desde el teclado deberemos usar la siguiente estructura.

&VariableTemporal

- El **&** indica que se va a leer desde teclado y **VariableTemporal** no es necesario declarar en la sección **DECLARE**, ya que se usa y se elimina.

- La VariableTemporal, guarda valores de tipo cadena debe usarse de la siguiente forma:

'&VariableTemporal'

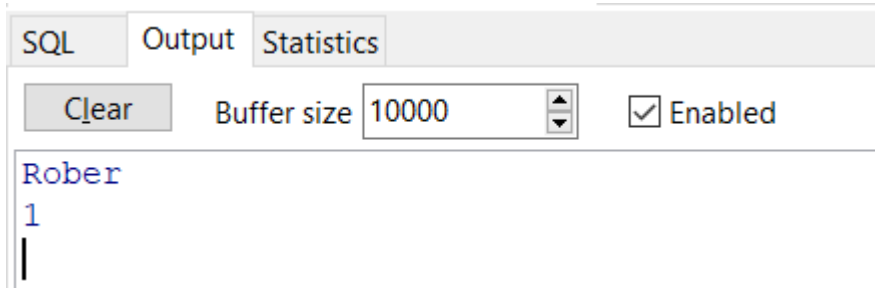
- **Ejemplos lectura de datos:**

Código

SQL | Output | Statistics

```
DECLARE
  depto emp.deptno%TYPE;
  nombre VARCHAR2(20);
BEGIN
  nombre := '&valorLeido';
  depto := &numero;

  DBMS_OUTPUT.PUT_LINE(nombre);
  DBMS_OUTPUT.PUT_LINE(depto);
END;
```

Resultado



- **Escritura de datos:** Para escribir datos por pantalla haremos uso de la función **PUT_LINE** del paquete **ORACLE DBMS_OUTPUT**.

```
DBMS_OUTPUT.PUT_LINE('Hola mundo');
```

- Si tenemos que concatenar valores de variables usaremos el operador **||**

```
DBMS_OUTPUT.PUT_LINE('Hola ' || nombre);
```

- **Ejemplo de escritura de datos:**

Código

SQL | Output | Statistics

```
DECLARE
  depto emp.deptno%TYPE;
  nombre VARCHAR2(20);
BEGIN
  SELECT emp.deptno INTO depto
  FROM emp emp
  WHERE emp.empno = 7369;

  DBMS_OUTPUT.PUT_LINE('El número del departamente es: ' || depto);
END;
```

Resultado

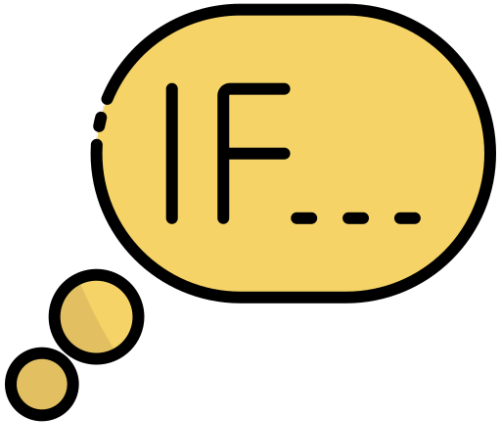
SQL | Output | Statistics

Clear | Buffer size 10000 | Enabled

```
El número del departamente es: 20
```

ESTRUCTURAS DE CONTROL DE FLUJO

- **Sentencia IF**



Opción 1

```
IF condición THEN
    <instrucciones>
END IF;
```

Opción 2

```
IF condición THEN
    <instrucciones>
ELSE
    <instrucciones>
END IF;
```

Opción 3

```
IF condición THEN
    <instrucciones>
ELSIF condición2 THEN
    <instrucciones>
ELSE
    <instrucciones>
END;
```

- **Sentencia CASE**

```
CASE selector/variable
    WHEN exprexión1 THEN
        <sentencias1>
    WHEN exprexión2 THEN
        <sentencias3>
    …..

    WHEN exprexiónN THEN
        <sentenciasN>
    [ELSE
        <sentenciasPorDefecto>]
END [CASE];
```

```
CASE
    WHEN condición1 THEN
        <sentencias1>
    WHEN condición2 THEN
        <sentencias3>
    …..

    WHEN condicionN THEN
        <sentenciasN>
    [ELSE
        <sentenciasPorDefecto>]
END CASE;
```

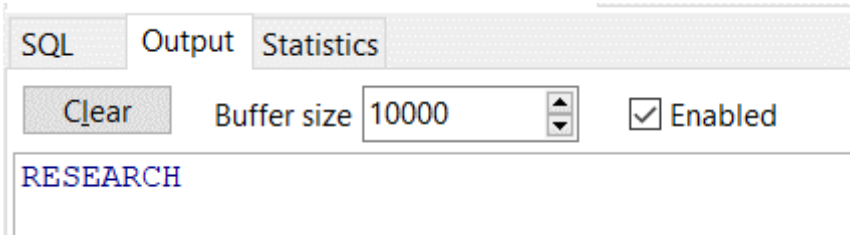
- **Ejemplo Case (Opción 1 V1)**

Código

SQL | Output | Statistics

```
DECLARE
  depto emp.deptno%TYPE;
  nombre VARCHAR2(20);
BEGIN
  SELECT emp.deptno INTO depto
  FROM emp emp
  WHERE emp.empno = 7369;

  nombre := CASE depto
              WHEN 10 THEN 'ACCOUNTING'
              WHEN 20 THEN 'RESEARCH'
              WHEN 80 THEN 'SALES'
              ELSE 'Otro departamento'
            END;

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

Resultado

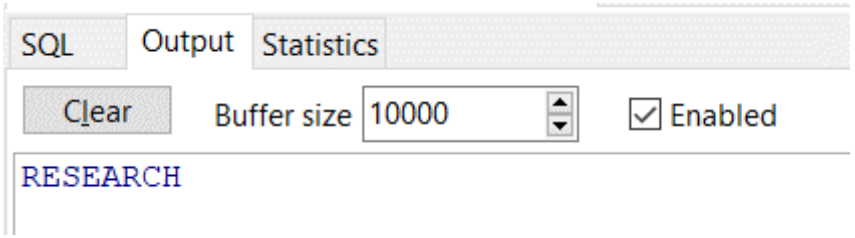


- **Ejemplo Case (Opción 1 V2)**

Código

```
DECLARE
  depto emp.deptno%TYPE;
  nombre VARCHAR2(20);
BEGIN
  SELECT CASE emp.deptno
              WHEN 10 THEN 'ACCOUNTING'
              WHEN 20 THEN 'RESEARCH'
              WHEN 80 THEN 'SALES'
              ELSE 'Otro departamento'
         END CASE
  INTO nombre
  FROM emp emp
  WHERE emp.empno = 7369;

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

Resultado



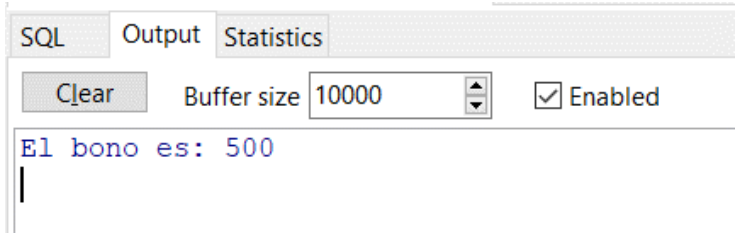
- **Ejemplo Case (Opción 2)**

Código

SQL | Output | Statistics

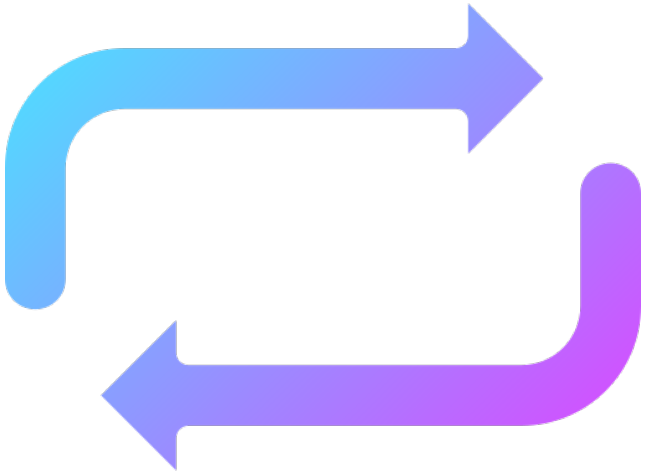
```
DECLARE
  sal REAL;
  bono NUMBER(5);
BEGIN
  SELECT emp.sal
  INTO sal
  FROM emp emp
  WHERE emp.empno = 7499;

  CASE
      WHEN sal < 5000 THEN
           bono := 500;
      WHEN sal > 12000 THEN
           bono := 100;
      ELSE Bono := 200;
   END CASE;


  DBMS_OUTPUT.PUT_LINE('El bono es: ' || bono);
END;
```

Resultado



- **Sentencia LOOP:** Los bucles **LOOP** ejecutan las instrucciones de manera repedita hasta que se cumpla la condición de salida indicada por **EXIT WHEN <condiciponSalida>.**

```
LOOP
    <sentencias>
    EXIT WHEN <condicionDeFin>;
END LOOP;
```

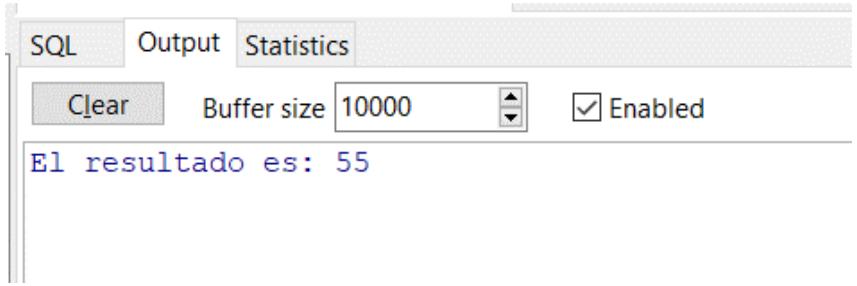


- **Ejemplo LOOP**

Código

```
DECLARE
  vari NUMBER(2) := 10;
  suma NUMBER(3) := 0;
BEGIN
  LOOP
        suma := suma + vari;
        vari := vari - 1;
        EXIT WHEN vari < 1;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE('El resultado es: ' || suma);
END;
```

Resultado

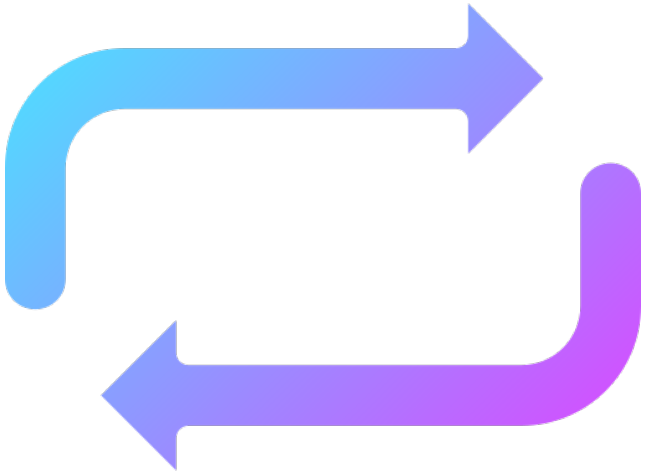


- **Sentencia WHILE:** Los bucles **WHILE** evalúan la condición de parada antes de iniciar el bucle.

WHILE <condicionDeFin> **LOOP**

<sentencias>

END LOOP;



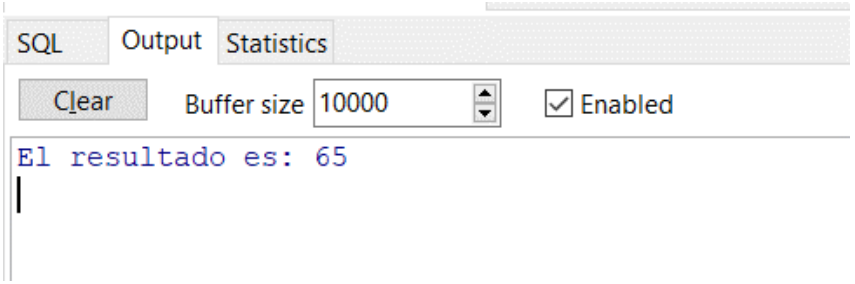
- **Ejemplo WHILE**

Código

SQL | Output | Statistics

```
DECLARE
  vari NUMBER(2) := 0;
  suma NUMBER(3) := 20;
BEGIN
  WHILE vari < 10 LOOP
        suma := suma + vari;
        vari := vari + 1;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE('El resultado es: ' || suma);
END;
```

Resultado

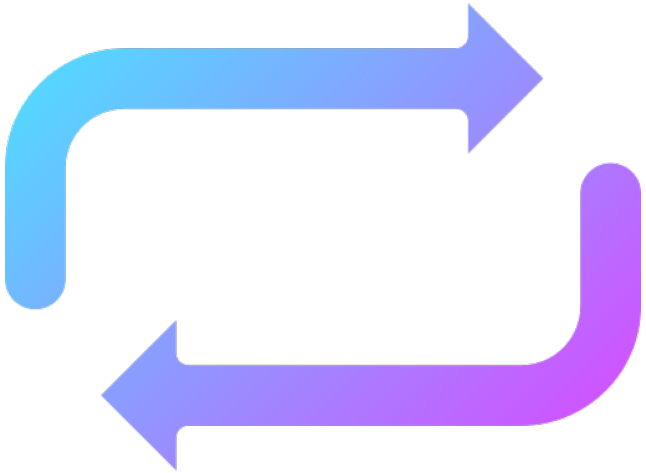


- **Sentencia FOR:** Los bucles **FOR** se ejecutan un número concreto de veces, el cual solemos saber. Se una variable auxiliar a como de contador, pero no podemos cambiar su valor dentro del mismo

FOR <varAux> **IN** <rango> **LOOP**

<sentencias>

END LOOP;



- Siempre se ejecuta de menos a más, pero si queremos ejecutar en orden inverso debemos usar la palabra reservada **REVERSE**, en la siguiente diapositiva se muestra cómo queda la declaración.

- **Ejemplo FOR**

Código

SQL | Output | Statistics

```
DECLARE
  vari NUMBER := 0;
BEGIN
  FOR numero IN 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE('Parcial: ' || numero);
    vari := vari + numero;
    DBMS_OUTPUT.PUT_LINE('Total: ' || vari);
    DBMS_OUTPUT.PUT_LINE('');
  END LOOP;
END;
```

Resultado

SQL | Output | Statistics

Clear | Buffer size 10000 | Enabled

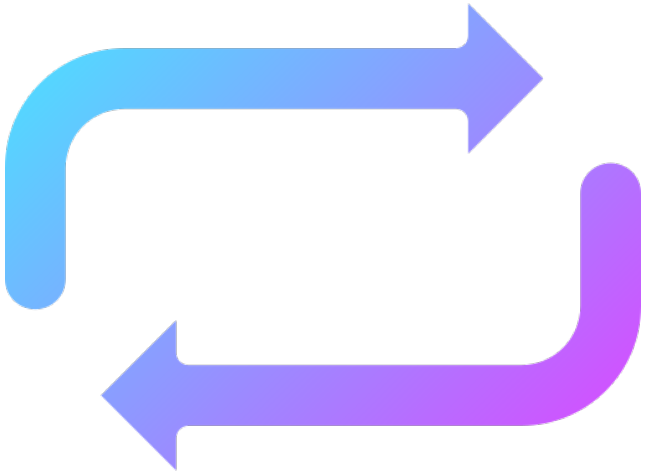
```
Parcial: 1
Total: 1

Parcial: 2
Total: 3

Parcial: 3
Total: 6

Parcial: 4
Total: 10

Parcial: 5
Total: 15
```

- **Sentencia FOR:** Los bucles **FOR** se ejecutan un número concreto de veces, el cual solemos saber. Se una variable auxiliar a como de contador, pero no podemos cambiar su valor dentro del mismo

FOR <varAux> **IN REVERSE** <rango> **LOOP**

<sentencias>

END LOOP;



- **Ejemplo FOR con REVERSE**

Código

SQL | Output | Statistics

```
DECLARE
  vari NUMBER := 0;
BEGIN
  FOR numero IN REVERSE 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE('Parcial: ' || numero);
    vari := vari + numero;
    DBMS_OUTPUT.PUT_LINE('Total: ' || vari);
    DBMS_OUTPUT.PUT_LINE('');
  END LOOP;
END;
```

Resultado

SQL | Output | Statistics

Clear | Buffer size 10000 | Enabled

```
Parcial: 10
Total: 10

Parcial: 9
Total: 19

Parcial: 8
Total: 27

Parcial: 7
Total: 34

Parcial: 6
Total: 40
```

PROCEDIMIENTOS Y FUNCIONES

- Los procedimientos y funciones almacenadas son conjuntos de comandos SQL que pueden ser almacenados en el servidor.

- Se asocia un nombre a un conjunto determinado de instrucciones para, posteriormente, ejecutarlo tantas veces como se desee sin necesidad de volver a escribirlas.

- La diferencia entre un procedimiento y una función es que las funciones tienen un tipo de retorno y los procedimientos no.

- En las siguientes diapositivas se muestra la sintaxis que tienen procedimientos y funciones

- Sintaxis de procedimientos:

CREATE [OR REPLACE] PROCEDURE <nombreProcedimiento>

[(nombreParametro **[IN | OUT | INOUT]** TIPO_DATO, …)] {**IS | AS**}

[PRAGMA AUTONOMOUS_TRANSACTION;]

BEGIN
<sentenciasProcedimiento>
[EXCEPTION]
<excepcionesDefinidas>
END <nombreProcedimiento>;

- Sintaxis de funciones:

```
CREATE [OR REPLACE] FUNCTION<nombreFuncion>
    [(nombreParametro [IN | OUT | INOUT] TIPO_DATO, …)]

    RETURN TIPO_DATO {IS | AS}

    [DETERMINISTIC]

    [PRAGMA AUTONOMOUS_TRANSACTION;]

BEGIN
    <sentenciasFuncion>
[EXCEPTION]
    <excepcionesDefinidas>
END <nombreFuncion>;
```

- La **PRAGMA AUTONOMOUS_TRANSACTION** marca el procedimiento como autónomo. Un procedimiento autónomo permite realizar **COMMIT** o **ROLLBACK** de las sentencias SQL propias sin afectar la transacción que lo haya llamado.

- No se usa la cláusula **DECLARE** puesto que va implícita en el **IS** o el **AS**. No existe diferencia entre el uso de uno u otro.

- **DETERMINISTIC** ayuda al optimizador, si una función fue ejecutada anteriormente y se vuelve a ejecutar con los mismos datos, con esta notación el optimizador devuelve el valor obtenido previamente ya que lo tiene almacenado en lugar de realizar la llamada nuevamente.

- Los parámetros pueden ser **IN** (entrada), **OUT** (salida), **INOUT**(ambos), en caso de no indicar uno, el valor por defecto es **IN**.

- Los procedimientos/funciones por si solos no hacen nada, hasta que no los invocamos en un bloque anónimo o en otro procedimiento/función o dentro de la lista de selección de una **SELECT.**

- Ejemplo de procedimientos:

SQL | Output | Statistics

```
CREATE OR REPLACE PROCEDURE ALTA_SAL_DEPT(P_NUMERO_DEPARTAMENTO IN DEPT.DEPTNO%TYPE) IS
CURSOR cCursor IS
      SELECT SUM(EMPE.SAL) SALARIO_DEPT,
             EMPE.DEPTNO DEPTNO
      FROM EMP EMPE
      WHERE EMPE.DEPTNO = P_NUMERO_DEPARTAMENTO
      GROUP BY EMPE.DEPTNO;
  cCursorInto cCursor%ROWTYPE;
BEGIN
  OPEN cCursor;
  LOOP
    FETCH cCursor INTO cCursorInto;
    EXIT WHEN cCursor%NOTFOUND;

    INSERT INTO salario_departamentos VALUES(cCursorInto.DEPTNO, cCursorInto.SALARIO_DEPT);

  END LOOP;
  CLOSE cCursor;
  COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No se encontraron datos');
END ALTA_SAL_DEPT;
```

- Ejemplo de funciones:

SQL | Output | Statistics

```
CREATE OR REPLACE FUNCTION SALARIO_MEDIO_DEPT(P_NUMERO_DEPARTAMENTO IN DEPT.DEPTNO%TYPE, P_JOB IN EMP.JOB%TYPE)
RETURN NUMBER IS
  v_salario_medio NUMBER;
BEGIN
  SELECT AVG(EMPE.SAL)
  INTO v_salario_medio
  FROM EMP EMPE
  WHERE EMPE.DEPTNO = P_NUMERO_DEPARTAMENTO
        AND EMPE.JOB = P_JOB;
  RETURN v_salario_medio;
EXCEPTION
  WHEN OTHERS THEN
    v_salario_medio := 0;
END SALARIO_MEDIO_DEPT;
```

- Efectos colaterales de las funciones.
  - Cuando se llama desde un **SELECT** o una sentencia **INSERT, UPDATE o DELETE** ejecutada paralelamente, una función no puede modificar ninguna tabla de la BBDD.
  - Cuando se llama desde una sentencia **INSERT, UPDATE o DELETE**, una función no puede consultar o modificar ninguna tabla que aparezca en la sentencia SQL.
  - Cuando se llama desde una sentencia **SELECT, INSERT, UPDATE o DELETE**, la función no puede ejecutar ningún comando transaccional **(COMMIT, ROLLBACK, etc),** un comando de sesión **(SET ROLE, etc)** o control de sistema **(ALTER SYSTEM)**. Tampoco puede realizar ningún comando DDL **(CREATE, DROP, etc)**
  - Si se quebrantan estas reglas se producirá un error en tiempo de ejecución.
  - Para eiar las reclas anteriores se puede utilizar la directiva de compilación **PRAGMA RESTRICT_REFERENCES** que indica a los paquetes las limitaciones que tienen las funciones.

- Ejemplos de invocaciones.

Invocación de procedimiento

SQL Output Statistics

```
DECLARE
  varNumDep NUMBER := 0;
BEGIN
  varNumDep := &numerodepartamento;
  ALTA_SAL_DEPT(varNumDep);
END;
/
```

Invocación de función

```
BEGIN
  varDept := &numeroDepartamento;
  varCurro :='&trabajo';
  varResultado := SALARIO_MEDIO_DEPT(varDept,varCurro);
  DBMS_OUTPUT.PUT_LINE('El salario medio del departamento ' || varDept || ' es: ' || varResultado);
END;
/
```

- Para **eliminar** procedimientos o funciones usamos la siguiente sintaxis:

DROP {PROCEDURE | FUNCTION} <nombreProcedimiento/función>

CONTROL DE ERRORES

- Como ocurre con muchos lenguajes de programación, Oracle también es capaz de **gestionar, mediante excepciones, los errores** que se puedan producir durante la ejecución de un fragmento de código en entornos transaccionales.
- En estos, si durante la ejecución de una transacción se produce algún fallo, es posible deshacer toda la operación para evitar inconsistencias en los datos.
- Para declarar una excepción usamos la palabra reservada **EXCEPTION** en la sección declarativa.
- Para lanzar una excepción definida por nosotros o predefinida usamos la palabra reservada **RAISE**.

- Para recuperar el código y mensaje de error usamos **SQLCODE** y **SQLERRM** respectivamente.

- Para asignar códigos de error a una excepción definida por usuario usamos **PRAGMA EXCEPTION_INIT(nombreExcepción, -numeroErrorOracle)**

- Para asignar un mensaje a una excepción definida por el usuario utilizamos **RAISE_APPLICATION_ERROR(numeroError, mensajeErro [, {TRUE | FALSE}]).**

Ejemplo creación y raise de excepciones.

SQL | Output | Statistics

```
DECLARE
  error1  EXCEPTION;
  cantidad NUMBER(4);
BEGIN
  IF cantidad = 50 THEN
     RAISE error1;
  END IF;

  IF cantidad < 0 THEN
     RAISE INVALID_NUMBER;
  END IF;
EXCEPTION
  WHEN error1 THEN
       NULL;
  WHEN INVALID_NUMBER THEN
       NULL;
END;
```

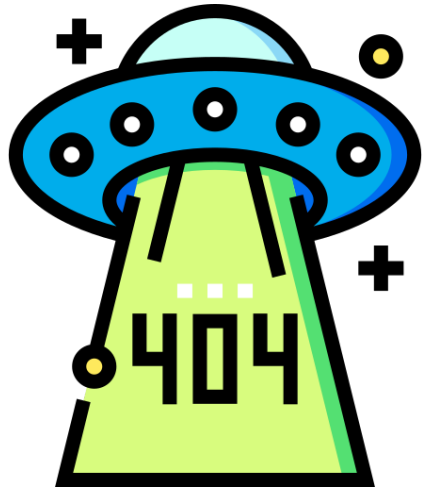
Ejemplo de asociación de códigos y mensajes de error a excepciones definidas.

```
DECLARE
  salarioAlto  EXCEPTION;
  salario NUMBER(8,2);
  PRAGMA EXCEPTION_INIT(salarioAlto, -20104);
BEGIN

  SELECT EMP.SAL
  INTO salario
  FROM EMP EMP
  WHERE EMP.EMPNO = 7499;

  IF salario > 2500 THEN
     RAISE_APPLICATION_ERROR(-20104, 'El salario es Alto');
  END IF;
EXCEPTION
  WHEN salarioAlto THEN
       DBMS_OUTPUT.PUT_LINE('Error: ' || SQLCODE || ' Mensaje: ' || SQLERRM);
END;
```

Ejemplo recuperación de código y mensaje de error .

SQL | Output | Statistics

```
DECLARE
  salarioAlto  EXCEPTION;
  salario NUMBER(8,2);
  PRAGMA EXCEPTION_INIT(salarioAlto, -20104);
  numeroError NUMBER;
  mensajeError VARCHAR2(100);
BEGIN

  SELECT EMP.SAL
  INTO salario
  FROM EMP EMP
  WHERE EMP.EMPNO = 7499;

  IF salario > 2500 THEN
     RAISE salarioAlto;
  END IF;
EXCEPTION
  WHEN OTHERS THEN
       numeroError := SQLCODE;
       mensajeError := SUBSTR(SQLERRM, 1, 100);
       DBMS_OUTPUT.PUT_LINE('Error: ' || numeroError || ' Mensaje: ' || mensajeError);
END;
```



- **Excepciones pre-defindas de ORACLE**

Nombre excepción	Error Oracle	Descripción
ACCESS_INTO_NULL	ORA-06530	Se intenta asignar valores a un atributo de un objeto que contiene nulos.
CASE_NOT_FOUND	ORA-06592	Ninguna de las opciones WHN de una sentencia CASE ha sido seleccionada y no existe la cláusula ELSE.
COLLECTION_IS_NULL	ORA-06531	Se intenta aplicar métodos de colección diferentes a EXIST a una tabla anidada o un Varray y esta contiene valores nulos o no está inicializada.
CURSOR_ALREADY_OPEN	ORA-06511	Se intenta abrir un cursor que ya está abierto.
DUP_VAL_ON_INDEX	ORA-00001	Se intenta guardar un valor duplicado en un índice que no permite valores duplicados.
INVALID_CURSOR	ORA-01001	Se intenta realizar una operación sobre un cursor que está cerrado.

- **Excepciones pre-defindas de ORACLE**

Nombre excepción	Error Oracle	Descripción
INVALID_NUMBER	ORA-01722	En un comando SQL la conversión de una cadena alfanumérica a un número es incorrecta ya que no representa un número válido. En PL/SQL levanta la excepción VALUE_ERROR.
LOGIN_DENIED	ORA-01017	Se intenta conectarse a Oracle con un usuario y una contraseña incorrectos.
NO_DATA_FOUND	ORA-01403	Un SELECT INTO no devuelve filas, o se referencia a un elemento borrado de una tabla anidad o un elemento no inicializado en una tabla indexada. Las funciones agregadas de grupo (AVG, SUM, COUNT, etc) siempre devuelven nulo o un cero por lo que un comando SELECT con funciones agregadas nunca levantará esta excepción. Un comando FETCH puede que no devuelva filas por lo que no levantará esta excepción en el caso de que no devuelva ninguna fila.

- **Excepciones pre-defindas de ORACLE**

Nombre excepción	Error Oracle	Descripción
NOT_LOGGED_ON	ORA_01012	Se intenta realizar una llamada a una base de datos sin estar conectado a ella.
PROGRAM_ERROR	ORA-06501	Se ha producido un error interno de PL/SQL.
ROWTYPE_MISMATCH	ORA-06504	La host variable y la variable de un cursosr PL/SQL no son del mismo tipo.
SELF_IS_NULL	ORA-30625	Se intenta usar el método MEMBER a una instancia nula.
STORAGE_ERROR	ORA-06500	Falta de recursos de memoria o está corrupta.
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Se intenta referenciar a un elemento de una tabla anidada o un VARRAY utilizando un valor mayor que el número de elementos de la tabla.

- **Excepciones pre-defindas de ORACLE**

Nombre excepción	Error Oracle	Descripción
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Se intenta referenciar a un elemento de una tabla anidada o un VARRAY utilizando a un valor que está fuera del rango permitido.
SYS_INVALID_ROWID	ORA-01410	La conversión de una cadena alfanumérica a un tipo ROWID universal es incorrecta porque no representa un valor valido
TIMEOUT_ON_RESOURCE	ORA-00051	Se ha producido un TIME-OUT esperando un recurso
TOO_MANY_ROWS	ORA-01422	La select recupera más filas de lo que se puede almacenar
VALUE_ERROR	ORA-06502	Se ha producido un error en una operación aritmética, conversión, truncamiento o límite de precisión
ZERO_DIVIDE	ORA-01476	Se ha producido una división por cero.

CURSORES

- Un **cursor**, es un objeto que hace referencia a un conjunto de datos obtenidos de una consulta.

- A través del cursor se pueden recorrer los datos obtenidos a través de la consulta, uno por línea.

- **CURSOR** sirve para definir un cursor.

- Los cursores deben abrirse (**OPEN**) antes de leerse los datos de ellos y se deben cerrar (**CLOSE**) una vez se ha terminado.

- **FETCH** extrae la siguiente fila de un cursor.

- **CLOSE** se cierra el cursor**.**

Definición de un cursor

CURSOR <nombreCursor> **IS** SELECT;

o

CURSOR <nombreCursor> **[(listaParametros)] IS** SELECT;

Apertura y cierre de un cursor

1 - Declaración.
2 - Abrir.
3 - Lectura.
4 - Cerrar Cursor.

OPEN <nombreCursor> [parametros];

CLOSE <nombreCursor>;

Lectura de un cursor

FETCH <nombreCursor> INTO <registroLectura>;

USAMOS **%ROWTYPE** PARA CREARLO

- **Atributos del cursor**

ATRIBUTO	DESCRIPCIÓN
%FOUND	Cuando se abre un cursor o un cursor variable, este atributo contiene el valor NULL. Cuando se ejecute el primer FETCH contendrá el valor FALSE si no se ha devuelto ninguna fila o TRUE si se ha devuelto una fila.
%ISOPEN	Devolverá el valor TRUE si el cursor ha sido abierto o por el contrario FALSE si el cursor no ha sido abierto.
%NOTFOUND	Es el atributo opuesto a %FOUND. Sera FALSE cuando se haya devuelto una fila en el FETCH o TRUE si no ha devuelto una fila. Contendrá el valor NULL si no se ha realizado ningún FECH.
%ROWCOUNT	Cuando un cursor o un cursor variable ha sido abierto, este atributo toma el valor cero. A medida que se van realizando FETCH el valor se va incrementando con el número de filas que devuelve cada FETCH.

- Ejemplo:

SQL Output Statistics

```
DECLARE
  CURSOR cCursor(numeroDepartamento IN NUMBER) IS
         SELECT DEPT.DEPTNO NUMERO_DEPARTAMENTO,
                DEPT.DNAME NOMBRE_DEPARTAMENTO,
                DEPT.LOC LOCALIZACION
         FROM DEPT DEPT
         WHERE DEPT.DEPTNO > numeroDepartamento;

  variableInto cCursor%ROWTYPE;
BEGIN
  OPEN cCursor(10);

  LOOP
    FETCH cCursor INTO variableInto;
    EXIT WHEN cCursor%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE('Codigo: ' || variableInto.NUMERO_DEPARTAMENTO ||
                         ' Nombre: ' || variableInto.NOMBRE_DEPARTAMENTO ||
                         ' Localización: ' || variableInto.LOCALIZACION);
  END LOOP;

  CLOSE cCursor;
EXCEPTION
  WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE('Error: ' || SQLCODE || ' Mensaje: ' || SQLERRM);
END;
```

- Ejemplo con for:

SQL | Output | Statistics

```
DECLARE
  CURSOR cCursor(numeroDepartamento IN NUMBER) IS
         SELECT DEPT.DEPTNO NUMERO_DEPARTAMENTO,
                DEPT.DNAME NOMBRE_DEPARTAMENTO,
                DEPT.LOC LOCALIZACION
         FROM DEPT DEPT
         WHERE DEPT.DEPTNO > numeroDepartamento;
BEGIN
  FOR reg_cursor IN cCursor(10) LOOP
    DBMS_OUTPUT.PUT_LINE('Codigo: ' || reg_cursor.NUMERO_DEPARTAMENTO ||
                         ' Nombre: ' || reg_cursor.NOMBRE_DEPARTAMENTO ||
                         ' Localización: ' || reg_cursor.LOCALIZACION);
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
       DBMS_OUTPUT.PUT_LINE('Error: ' || SQLCODE || ' Mensaje: ' || SQLERRM);
END;
```

DISPARADORES (TRIGGERS)

- Los **disparadores o triggers** son procedimientos de la Base de Datos que se ejecutan o activan cada vez que ocurre un evento determinado sobre una tabla determinada, según se haya indicado en el momento de su implementación.

- Los eventos que se pueden asociar a la ejecución de un TRIGGER son: **INSERT, UPDATE y DELETE.**

- También puede decidirse que se activen antes o después del evento en cuestión, utilizando las palabras reservadas **BEFORE** (antes) y **AFTER** (después) para indicar que el disparador se ejecute antes o después que la sentencia que lo activa.

- Las palabras :**NEW** y :**OLD** se emplean para referirse a las filas afectadas por el disparador, es decir, **a las filas de la tabla sobre la que se activa**, para referirse al estado de esa fila, **antes (:OLD)** o **después (:NEW)** de haber actuado el disparador.

- En un disparador **INSERT** sólo se podrá utilizar la palabra :**NEW** puesto que no hay versión anterior de esa fila, en un disparador **UPDATE** se podrá acceder a cada fila **antes (:OLD) y después (:NEW)** de haber sido actualizada; y en un disparador **DELETE** sólo se puede utilizar la palabra :**OLD** puesto que no hay nueva versión de la fila.

- Hay que tener en cuenta que toda columna **precedida por :OLD es de sólo lectura**, por lo que se podrá hacer referencia a ella, pero sólo para leerla. En el caso de las columnas precedidas por :**NEW**, se **podrá leer y también modificar su valor** con la instrucción **SET**.

- Declaración TRIGGER.

CREATE [OR REPLACE] TRIGGER [esquema.]<nombreTrigger>
{BEFORE | AFTER} sucesoDisparo **ON** <nombreTabla>

[**FOR EACH ROW** [**WHEN** condicionDisparo]]

DECLARE
    <declaraciones>
BEGIN
    <sentencias>
END <nombreTrigger>;

```
CREATE OR REPLACE TRIGGER aumentoSalario
       AFTER UPDATE ON EMP
       FOR EACH ROW
DECLARE
       dif NUMBER(8,2);
BEGIN
       dif := :NEW.sal - :OLD.sal;
       DBMS_OUTPUT.PUT_LINE('Diferencia de salario: ' || dif);
END aumentoSalario;
```

- Por ejemplo, el disparador **aumentoSalario**, se activará una vez por cada fila (**FOR EACH ROW**) actualizada (**UPDATE**) en la tabla emp. Y concretamente después de que éstas se actualicen (**AFTER**).

- **Usos de los disparadores**

- Antes de comenzar a utilizar los disparadores, conviene conocer cuándo deben ser utilizados, y cuáles son sus limitaciones.

- Uno de los usos más comunes de los disparadores es el utilizarlos para realizar validaciones y para mantener actualizados los campos calculados, de manera que cuando ocurra algún cambio en los datos se pueda actualizar automáticamente dicho campo calculado, si tuviera que verse afectado.

- Además, permiten realizar tareas de auditoría, puesto que es posible registrar la actividad que ocurre en una o varias tablas en otra tabla, con el fin de registrar las operaciones que se realizan sobre ella, cuando se hacen, quién las hace, etc.

- **Limitaciones de los disparadores**

- En cuanto a las limitaciones, no puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia.

- Por ejemplo, no se pueden tener dos disparadores BEFORE UPDATE. Pero sí es posible tener los disparadores BEFORE UPDATE y BEFORE INSERT o BEFORE UPDATE y AFTER UPDATE.

- También cabe destacar que desde un disparador no es posible invocar a una consulta. Se puede llamar a procedimientos almacenados que devuelvan información a través de sus parámetros de salida siempre y cuando éste no realice dentro ninguna consulta.

- **Tablas mutantes**

- Tabla mutante es una tabla que está siendo modificada por una sentencia SQL (insert, update, delete) o por el efecto de un DELETE CASCADE asociado a la sentencia SQL. Restricciones sobre tablas mutantes:
  - La acción de una regla de tipo FOR EACH ROW no puede consultar ni actualizar una tabla mutante para un evento.
  - La acción de una regla de tipo FOR EACH STATEMENT activada como efecto de un DELETE CASCADE no puede consultar ni actualizar una tabla mutante para su evento.

PAQUETES

- Un paquete es una agrupación lógica de variables, constantes, tipos de datos y subprogramas PL/SQL (procedimientos y funciones).

- Los paquetes se dividen en la **especificación** y el **cuerpo:**

  - **Especificación:** Es la zona de declaración de variables, tipos, constantes, excepciones, cursores y subprogramas. NO CONTIENEN IMPLEMENTACIÓN.

  - **Cuerpo:** Zona en la que se implementa el código de los cursores y subprogramas definidos en la especificación, también puede contener otras declaraciones y otros subprogramas que no estén definidos en la especificación.

Declaración de la especificación:

```
CREATE [OR REPLACE] PACKAGE <nombrePaquete> {IS | AS}
    [Definición tipo_colección]
    [Definición tipo_registro]
    [Definición tipo_subtips]
    [Definición tipo_constantes]
    [Definición tipo_excepción]
    [Definición tipo_colección]
    [Especificación procedimiento]
    [Especificación función]
END <nombrePaquete>;
```

Declaración del cuerpo:

```
CREATE [OR REPLACE] PACKAGE <nombrePaquete> {IS | AS}
    [Definición tipo_colección]
    [Definición tipo_registro]
    [Definición tipo_subtips]
    [Definición tipo_constantes]
    [Definición tipo_excepción]
    [Definición tipo_colección]
    [Especificación procedimiento]
    [Especificación función]
BEGIN
    Sentencias procedurales
END <nombrePaquete>;
```

Ejemplo de declaración de especificación:

SQL | Output | Statistics

```
CREATE OR REPLACE PACKAGE GESTION_DE_EMPLEADOS AS
  PROCEDURE ALTA_EMP(P_NOMBRE IN EMP.ENAME%TYPE, P_JOB IN EMP.JOB%TYPE, P_BOSS IN EMP.MGR%TYPE);
  PROCEDURE BAJA_EMP(P_NUM_EMP IN EMP.EMPNO%TYPE);
  PROCEDURE MOD_EMP(P_NUM_EMP IN EMP.EMPNO%TYPE, P_DEPNO IN EMP.DEPTNO%TYPE);
END GESTION_DE_EMPLEADOS;
```

Ejemplo de declaración de cuerpo de paquete:

SQL Output Statistics

```
CREATE OR REPLACE PACKAGE BODY GESTION_DE_EMPLEADOS AS
  PROCEDURE ALTA_DEPT_INT(P_NUM_DEPT IN DEPT.DEPTNO%TYPE, P_NOMBRE_DEPAR IN DEPT.DNAME%TYPE, P_LOCA IN DEPT.LOC%

  PROCEDURE ALTA_EMP(P_NOMBRE IN EMP.ENAME%TYPE, P_JOB IN EMP.JOB%TYPE, P_BOSS IN EMP.MGR%TYPE) IS
    v_ultimo_num_emp EMP.EMPNO%TYPE;
    v_salario EMP.SAL%TYPE;
    v_departamentoBoss EMP.DEPTNO%TYPE;
    v_comision EMP.COMM%TYPE;
  BEGIN
    -- Obtenemos el último número disponible
    SELECT MAX(EMPE.EMPNO)
    INTO v_ultimo_num_emp
    FROM EMP EMPE;

    --Obtemeos el salario del nuevo empleado
    SELECT AVG(EMPE.SAL)
    INTO v_salario
    FROM EMP EMPE;

    SELECT EMPE.DEPTNO
    INTO v_departamentoBoss
    FROM EMP EMPE
    WHERE EMPE.EMPNO = P_BOSS;

    INSERT INTO EMP VALUES((v_ultimo_num_emp + 1) , P_NOMBRE, P_JOB, P_BOSS,SYSDATE,
    v_salario, v_comision, v_departamentoBoss);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error en la insercción: ' || SUBSTR(SQLERRM, 1, 200));
  END ALTA_EMP;
```

Invocación de funciones/procedimientos de un paquete:

```
NOMBRE_PAQUETE.NOMBRE_FUNCION(listaParam);
```

```
NOMBRE_PAQUETE.NOMBRE_PRODECIMIENTO(listaParam);
```

Ejemplo de llamada a un paquete:

SQL | Output | Statistics

```
DECLARE
  v_nombre_emp EMP.ENAME%TYPE;
  v_job EMP.JOB%TYPE;
  v_boss EMP.MGR%TYPE;

  --Variables para modificacion
  v_num_emp_up EMP.EMPNO%TYPE;
  v_num_dep_up EMP.DEPTNO%TYPE;

  --variable borrado
  v_num_emp_del EMP.EMPNO%TYPE;
BEGIN
  v_nombre_emp := '&nombreEmpleado';
  v_job := '&trabajoEmpleado';
  v_boss := &identificadorEmpleado;

  GESTION_DE_EMPLEADOS.ALTA_EMP(v_nombre_emp, v_job, v_boss);
END;
```

Muchas gracias por vuestra atención