

Ejercicios PL/SQL

Introducción

We will first create a provisional department, to which we assign the employees of department 20 before deleting said department. The program also reports the number of affected employees.

1 En el siguiente ejemplo se borra el departamento número 20, pero evitando posibles errores por violar restricciones de integridad referencial, pues el departamento tiene empleados asociados. Para ello crearemos antes un departamento provisional, al que asignamos los empleados del departamento 20 antes de borrar dicho departamento. El programa también informa del número de empleados afectados.

Antes de introducir el ejemplo deberemos introducir la instrucción SET SERVEROUTPUT ON o utilizar el menú de configuración de SQL*Plus para que muestre los mensajes.

```
DECLARE
    v_num_empleados NUMBER(2);
BEGIN
    INSERT INTO depart VALUES (99, 'PROVISIONAL', NULL);
    UPDATE emple SET dept_no = 99
        WHERE dept_no = 20;
    v_num_empleados := SQL%ROWCOUNT;
    DELETE FROM depart
        WHERE dept_no = 20;
    DBMS_OUTPUT.PUT_LINE(v_num_empleados ||
        ' Empleados ubicados en PROVISIONAL');
```

(Continuación)

```
EXCEPTION
    WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20000, 'Error en aplicación');
END;
/
```

El resultado de la ejecución del programa será:

5 Empleados ubicados en PROVISIONAL

Procedimiento PL/SQL terminado con éxito.

The next block displays the last name and occupation of the employee whose number is 7900.



El siguiente bloque visualiza el apellido y el oficio del empleado cuyo número es 7900.

```
DECLARE
  v_ape VARCHAR2(10);
  v_oficio VARCHAR2(10);
BEGIN
  SELECT apellido, oficio INTO v_ape, v_oficio
    FROM EMPLE WHERE EMP_NO = 7900;
  DBMS_OUTPUT.PUT_LINE(v_ape||'*'||v_oficio);
END;
/
```

El resultado de la ejecución del programa será:

```
JIMENO*EMPLEADO
Procedimiento PL/SQL terminado con éxito.
```

The example above, with exception handling.

3 El ejemplo anterior, con gestión de excepciones, sería:

```
DECLARE
  v_ape VARCHAR2(10);
  v_oficio VARCHAR2(10);
BEGIN
  SELECT apellido, oficio INTO v_ape, v_oficio
  FROM EMPLE WHERE EMP_NO = 7900;
  DBMS_OUTPUT.PUT_LINE(v_ape||' '||v_oficio);
```

(Continúa)

222

10. Introducción al lenguaje PL/SQL

10.2 Características del lenguaje

(Continuación)

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20000, 'ERROR no hay datos');
  WHEN TOO_MANY_ROWS THEN
    RAISE_APPLICATION_ERROR(-20000, 'ERROR demasiados datos');
  WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20000, 'Error en la aplicación');
END;
```

Create a trigger that will be executed automatically when an employee is deleted in the corresponding table, displaying the number and name of the deleted employees.

4 El siguiente código crea un *trigger* que se ejecutará automáticamente cuando se elimine algún empleado en la tabla correspondiente visualizando el número y el nombre de los empleados borrados:

```
CREATE OR REPLACE TRIGGER audit_borrado_emple
  BEFORE DELETE
  ON emple
  FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('BORRADO EMPLEADO'
    || ' ' || :old.emp_no
    || ' ' || :old.apellido);
END;
```

The following program will request the entry of a customer number and will display the name of the customer corresponding to the number entered. To enter the customer number we will use the SQL
*Plus substitution variables.

- 5 El siguiente programa solicitará la introducción de un número de cliente y visualizará el nombre del cliente correspondiente con el número introducido. Para introducir el número de cliente recurriremos a las variables de sustitución de SQL*Plus.

```
SQL> DECLARE
2     v_nom CLIENTES.NOMBRE%TYPE; --(ejemplo uso %TYPE)
3 BEGIN
4     SELECT nombre INTO v_nom
5     FROM clientes
6     WHERE CLIENTE_NO=&vn_cli;
7     DBMS_OUTPUT.PUT_LINE(v_nom);
8 END;
9
SQL>
```

Para ejecutar el programa utilizaremos RUN o /. Solicitará un valor para la variable de sustitución, y una vez introducido el valor sustituirá a la variable y se enviará el bloque al servidor para su ejecución.

```
SQL> /

Introduzca valor para vn_cli: 102
antiguo 6:      WHERE CLIENTE_NO=&vn_cli;
nuevo    6:      WHERE CLIENTE_NO=102;
LOGITRONICA S.L
```

Procedimiento PL/SQL terminado con éxito.

Introducing these lines from the SQL Plus prompt we will have a simple PL/SQL procedure to query a customer's data.

- 6 Introduciendo estas líneas desde el indicador de SQL*Plus dispondremos de un procedimiento PL/SQL sencillo para consultar los datos de un cliente:

```
SQL> CREATE OR REPLACE
2 PROCEDURE ver_depart (numdepart NUMBER)
3 AS
4     v_dnombre VARCHAR2(14);
5     v_localidad VARCHAR2(14);
6 BEGIN
7     SELECT dnombre, loc INTO v_dnombre, v_localidad
8     FROM depart
9     WHERE dept_no = numdepart;
10    DBMS_OUTPUT.PUT_LINE('Num depart: '||numdepart|| ' * Nombre dep: '|| v_dnombre ||
11                          ' * Localidad: '||v_localidad);
12 EXCEPTION
13     WHEN NO_DATA_FOUND THEN
14         DBMS_OUTPUT.PUT_LINE('No encontrado departamento ');
15 END ver_depart;
16 /
```

Procedimiento creado.

1. The price of a product whose number is passed as a parameter.
2. We will write a procedure that modifies the price of a product by passing it the product number and the new price. The procedure will verify that the price variation does not exceed 20%.
3. We will write a function that returns the value with VAT of an amount that will be passed as the first parameter. The function can also collect a second optional parameter, which will be the VAT rate, with the default value being 16.

7 Ejemplos de aplicación:

1. En el siguiente procedimiento se visualiza el precio de un producto cuyo número se pasa como parámetro.

```
SQL> CREATE OR REPLACE
  2  PROCEDURE ver_precio(v_num_producto NUMBER)
  3  AS
  4  v_precio NUMBER;
  5  BEGIN
  6  SELECT precio_actual INTO v_precio
  7  FROM productos
  8  WHERE producto_no = v_num_producto;
  9  DBMS_OUTPUT.PUT_LINE('Precio = '||v_precio);
 10  END;
 11  /
```

Procedimiento creado.

```
SQL> EXECUTE VER_PRECIO(50);
      Precio = 1050
```

Procedimiento PL/SQL terminado con éxito.

2. Escribiremos un procedimiento que modifique el precio de un producto pasándole el número del producto y el nuevo precio. El procedimiento comprobará que la variación de precio no supere el 20 por 100:

```
SQL> CREATE OR REPLACE
  2  PROCEDURE modificar_precio_producto
  3  (numproducto NUMBER, nuevoprecio NUMBER)
  4  AS
  5  v_precioant NUMBER(5);
  6  BEGIN
  7  SELECT precio_actual INTO v_precioant
  8  FROM productos
  9  WHERE producto_no = numproducto;
```

(Continuación)

```
10 IF (v_precioant * 0.20) > (nuevoprecio - v_precioant) THEN
11     UPDATE productos SET precio_actual = nuevoprecio
12     WHERE producto_no = numproducto;
13 ELSE
14     DBMS_OUTPUT.PUT_LINE('Error, modificación supera 20%');
15 END IF;
16
17 EXCEPTION
18     WHEN NO_DATA_FOUND THEN
19     DBMS_OUTPUT.PUT_LINE('No encontrado producto ' || numproducto);
20 END modificar_precio_producto;
21 /
```

Ejemplos de ejecución:

```
SQL> SET SERVEROUTPUT ON
```

```
SQL> EXECUTE MODIFICAR_PRECIO_PRODUCTO(60,300)
Procedimiento PL/SQL terminado con éxito.
```

```
SQL> SELECT PRECIO_UNI FROM PRODUCTOS WHERE COD_PRODUCTO=60;
PRECIO_UNI
-----
        300
```

```
SQL> EXECUTE MODIFICAR_PRECIO_PRODUCTO(60,10000)
Error, modificación supera 20%
Procedimiento PL/SQL terminado con éxito.
```

```
SQL> SELECT PRECIO_UNI FROM PRODUCTOS WHERE COD_PRODUCTO=3;
PRECIO_UNI
-----
        300
```

Observamos que en el segundo caso no se ha producido la modificación deseada. Aun así, el procedimiento ha terminado con éxito, ya que no se ha producido ningún error no tratado.

3. Escribiremos una función que devuelva el valor con IVA de una cantidad que se pasará como primer parámetro. La función también podrá recoger un segundo parámetro opcional, que será el tipo de IVA siendo el valor por defecto 16.

```
SQL> CREATE OR REPLACE FUNCTION con_iva (
2     cantidad NUMBER,
3     tipo NUMBER DEFAULT 16)
4
5     RETURN NUMBER
6 AS
7     v_resultado NUMBER (10,2) DEFAULT 0;
8 BEGIN
```

(Continúa)

(Continuación)

```
9      v_resultado := cantidad * (1 + (tipo / 100));
10      RETURN(v_resultado);
11 END con_iva;
12 /
```

Función creada.

Ahora podemos usar la función creada para realizar cálculos dentro de un bloque o programa PL/SQL:

```
SQL> BEGIN DBMS_OUTPUT.PUT_LINE(con_iva(200)); END;
2 /
232
```

Debemos recordar que hay que HACER ALGO con el valor devuelto por la función: visualizarlo, usarlo como parte de una expresión, etcétera. También podemos usarla en instrucciones SQL:

```
SQL> SELECT producto_no, precio_actual, con_iva(precio_actual) FROM productos;
```

PRODUCTO_NO	PRECIO_ACTUAL	CON_IVA(PRECIO_ACTUAL)
10	550	638
20	670	777,2
30	460	533,6
40	340	394,4
50	1050	1218
60	280	324,8
70	450	522
80	550	638

Let us suppose that we intend to modify the salary of a specified employee based on the number of employees that he is in charge of.

1 Supongamos que pretendemos modificar el salario de un empleado especificado en función del número de empleados que tiene a su cargo:

- Si no tiene ningún empleado a su cargo la subida será 50 €.
- Si tiene 1 empleado la subida será 80 €.
- Si tiene 2 empleados la subida será 100 €.
- Si tiene más de tres empleados la subida será 110 €.

Además, si el empleado es PRESIDENTE se incrementará el salario en 30 €.

DECLARE

v_empleado_no NUMBER(4,0);

-- emple al que subir salario

v_c_empleados NUMBER(2);

-- cantidad empl dependen de él

v_aumento NUMBER(7) DEFAULT 0;

-- importe que vamos a aumentar.

v_oficio VARCHAR2(10);

:


```

BEGIN
    v_empleado_no := &vt_empno;    -- var de sustitución lee n*emple

    SELECT oficio INTO v_oficio FROM emple
        WHERE emp_no = v_empleado_no;

    IF v_oficio = 'PRESIDENTE' THEN -- alternativa simple
        v_aumento := 30;
    END IF;

    SELECT COUNT(*) INTO v_c_empleados FROM emple
        WHERE dir = v_empleado_no;

    IF v_c_empleados = 0 THEN        -- alternativa múltiple
        v_aumento := v_aumento + 50;
    ELSIF v_c_empleados = 1 THEN
        v_aumento := v_aumento + 80;
    ELSIF v_c_empleados = 2 THEN
        v_aumento := v_aumento + 100;
    ELSE
        v_aumento := v_aumento + 110;
    END IF;

    UPDATE emple SET salario = salario + v_aumento WHERE emp_no = v_empleado_no;
    DBMS_OUTPUT.PUT_LINE(v_aumento);
END;
/

```

El resultado de la ejecución será:

```

Introduzca valor para vt_empno: 7839
140

```

En el programa anterior hemos utilizado una estructura ELSIF pero podíamos haber utilizado una estructura CASE en cualquiera de sus dos formatos:

CON CASE DE BÚSQUEDA

```

-----
CASE
WHEN v_c_empleados = 0 THEN
    v_aumento := v_aumento + 50;
WHEN v_c_empleados = 1 THEN
    v_aumento := v_aumento + 80;
WHEN v_c_empleados = 2 THEN
    v_aumento := v_aumento + 100;
ELSE
    v_aumento := v_aumento + 110;
END CASE;

```

CON CASE DE COMPROBACIÓN

```

-----
CASE v_c_empleados
WHEN 0 THEN
    v_aumento := v_aumento + 50;
WHEN 1 THEN
    v_aumento := v_aumento + 80;
WHEN 2 THEN
    v_aumento := v_aumento + 100;
ELSE
    v_aumento := v_aumento + 110;
END CASE;

```

Suppose we want to parse a string containing both last names to store the first last name in a variable we'll call v_1name. We understand that the first surname ends when we encounter any character other than alphabetic ones.



Caso práctico

- 2 Supongamos que deseamos analizar una cadena que contiene los dos apellidos para guardar el primer apellido en una variable a la que llamaremos `v_1apel`. Entendemos que el primer apellido termina cuando encontramos cualquier carácter distinto de los alfabéticos (en mayúsculas).

```
DECLARE
  v_apellidos VARCHAR2(25);
  v_1apel VARCHAR2(25);
  v_caracter CHAR;
  v_posicion INTEGER :=1;
```

(Continúa)

(Continuación)

```
BEGIN
  v_apellidos := '&vs_apellidos';

  v_caracter := SUBSTR(v_apellidos,v_posicion,1);
  WHILE v_caracter BETWEEN 'A' AND 'Z' LOOP
    v_lapel := v_lapel || v_caracter;
    v_posicion := v_posicion + 1;
    v_caracter := SUBSTR(v_apellidos,v_posicion,1);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('1er Apellido: '||v_lapel||'*');
END;
/
```

El resultado de la ejecución será:

```
Introduzca valor para vs_apellidos: GIL MORENO
1er Apellido:GIL*
Procedimiento PL/SQL terminado con éxito.
```

El mismo ejemplo con un bucle LOOP... END LOOP será:

```
DECLARE
  v_apellidos VARCHAR2(25);
  v_lapel VARCHAR2(25);
  v_caracter CHAR;
  v_posicion INTEGER :=1;
BEGIN
  v_apellidos := '&vs_apellidos';

  -- desaparece la asignación de v_caracter antes del bucle
  -- se asignará dentro al comienzo del bucle.
  LOOP
    v_caracter := SUBSTR(v_apellidos,v_posicion,1);
    EXIT WHEN v_caracter NOT BETWEEN 'A' AND 'Z';
    v_lapel := v_lapel || v_caracter;
    v_posicion := v_posicion + 1;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('1er Apellido: '||v_lapel||'*');
END;
/
```

We are going to build a PL/SQL block that writes the string 'HELLO' backwards in two ways

3 Vamos a construir de dos maneras un bloque PL/SQL que escriba la cadena 'HOLA' al revés.

Utilizando un bucle FOR	Utilizando un bucle WHILE
<pre>SQL> DECLARE 2 r_cadena VARCHAR2(10); 3 BEGIN 4 FOR i IN REVERSE 1..LENGTH('HOLA') LOOP 5 r_cadena := r_cadena SUBSTR('HOLA',i,1); 6 END LOOP; 7 DBMS_OUTPUT.PUT_LINE(r_cadena); 8* END; SQL> / ALOH Procedimiento PL/SQL terminado con éxito.</pre>	<pre>SQL> DECLARE 2 r_cadena VARCHAR2(10); 3 i BINARY_INTEGER; 4 BEGIN 5 i := LENGTH('HOLA'); 6 WHILE i >= 1 LOOP 7 r_cadena=r_cadena SUBSTR('HOLA',i,1); 8 i := i - 1; 9 END LOOP; 10 DBMS_OUTPUT.PUT_LINE(r_cadena); 11* END; SQL> / ALOH Procedimiento PL/SQL terminado con éxito.</pre>

We will create a procedure that receives an employee number and a string corresponding to his new job. The procedure must locate the employee, modify the job and display the changes made.

Caso práctico

4 Crearemos un procedimiento que reciba un número de empleado y una cadena correspondiente a su nuevo oficio. El procedimiento deberá localizar al empleado, modificar el oficio y visualizar los cambios realizados.

```
CREATE OR REPLACE PROCEDURE cambiar_oficio (
  num_empleado NUMBER,      -- En los parámetros ..
  nuevo_oficio VARCHAR2)    -- ..no se especifica tamaño
AS
  v_anterior_oficio emp.oficio%TYPE;
BEGIN
  SELECT oficio INTO v_anterior_oficio FROM emp
    WHERE emp_no = num_empleado;

  UPDATE emp SET oficio = nuevo_oficio
    WHERE emp_no = num_empleado;
  DBMS_OUTPUT.PUT_LINE(num_empleado||'*Oficio Anterior: '||v_anterior_oficio||
    '*Oficio Nuevo   : '||nuevo_oficio);

END cambiar_oficio;
/
```

El sistema responderá:

Procedimiento creado.

Ahora el procedimiento está creado y almacenado en la base de datos. Para ejecutarlo podemos invocar el procedimiento desde cualquier herramienta de Oracle, por ejemplo, desde SQL*Plus:

```
SQL> EXECUTE CAMBIAR_OFICIO(7902, 'DIRECTOR');
7902*Oficio Anterior:ANALISTA*Oficio Nuevo   :DIRECTOR
Procedimiento PL/SQL terminado con éxito.
```

Suppose that we have been requested a foreign exchange program for a bank that meets the following specifications.

- You will receive an amount in euros and the change (currency/euros) of the currency.

- You may also receive an amount corresponding to the commission that will be charged for the transaction. In the event that you do not receive said amount, the program will calculate the commission, which will be 0.2% of the amount, with a minimum of 3 euros.
- The program will calculate the commission, deduct it from the initial amount and calculate the change in the desired currency, returning these two values (commission and change) to the current parameters of the program that makes the call to request the currency exchange.

(Continuación)

```
END IF;  
cantidad_divisas := (cantidad_euros - cantidad_comision) * cambio_actual;  
END;  
/
```

Una vez creado el procedimiento podremos diseñar programas que hagan uso de él teniendo en cuenta que los parámetros formales para llamar al programa deberán ser cuatro. De éstos, los dos últimos deberán ser variables, que recibirán los valores de la ejecución del programa, tal como aparece en el siguiente procedimiento:

```
CREATE OR REPLACE PROCEDURE mostrar_cambio_divisas (  
    eur NUMBER,  
    cambio NUMBER)  
AS  
    v_comision NUMBER (9);  
    v_divisas NUMBER (9);  
BEGIN  
    Cambiar_divisas(eur, cambio, v_comision, v_divisas);  
    DBMS_OUTPUT.PUT_LINE ('Euros          : ||'  
        TO_CHAR( Eur, '999,999,999.999'));  
    DBMS_OUTPUT.PUT_LINE ('Divisas X 1 euro : ||'  
        TO_CHAR( cambio, '999,999,999.999'));  
    DBMS_OUTPUT.PUT_LINE ('Euros Comisión  : ||'  
        TO_CHAR( v_comision, '999,999,999.999'));  
    DBMS_OUTPUT.PUT_LINE ('Cantidad divisas : ||'  
        TO_CHAR( v_divisas, '999,999,999.999'));  
END;  
/
```

Llamamos al programa pasándole la cantidad y el cambio respecto al euro de la divisa queremos cambiar a euros.

```
SQL> EXECUTE MOSTRAR_CAMBIO_DIVISAS(2500, 1.220);  
Euros          :          2,500.000  
Divisas X 1 euro :          1.220  
Euros Comisión  :          5.000  
Cantidad divisas :        3,044.000
```

Procedimiento PL/SQL terminado con éxito.

EXCEPCIONES

The following example illustrates what we have seen so far regarding cursors and cursor attributes: it is about displaying the last names of the employees belonging to department 20 by numbering them sequentially.

- 1 El siguiente ejemplo ilustra lo que hemos visto hasta ahora respecto a cursores y atributos de cursor: se trata de visualizar los apellidos de los empleados pertenecientes al departamento 20 numerándolos secuencialmente.

```
DECLARE
  CURSOR c1 IS
    SELECT apellido FROM emple WHERE dept_no=20;
  v_apellido VARCHAR2(10);
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO v_apellido;
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(c1%ROWCOUNT, '99.')
      || v_apellido);
    EXIT WHEN c1%NOTFOUND;
  END LOOP;
  CLOSE c1;
END;
```

El resultado de la ejecución será:

```
1.SANCHEZ
2.JIMENEZ
3.GIL
4.ALONSO
5.FERNANDEZ
5.FERNANDEZ
```

En este ejemplo observamos que el último FETCH no devuelve ningún valor, no incrementa el atributo %ROWCOUNT y no sobrescribe el valor de la variable del cursor. Es evidente que el programa está mal diseñado, ya que procesa (visualiza) la información supuestamente recuperada antes de comprobar si se ha recuperado información nueva.

En caso de que FETCH no recupere una nueva fila:

- No se incrementará el atributo %ROWCOUNT.
- No se sobrescribe el valor de la variable del cursor.

The following example displays the employees of any department using coupling variables.

2 En el siguiente ejemplo se visualizan los empleados de un departamento cualquiera usando variables de acoplamiento:

```
CREATE OR REPLACE PROCEDURE ver_emple_por_dept (  
  dep VARCHAR2)  
AS  
  v_dept NUMBER(2);  
  CURSOR c1 IS  
    SELECT apellido FROM emple WHERE dept_no = v_dept;  
  v_apellido VARCHAR2(10);  
BEGIN  
  v_dept := dep;  
  OPEN c1;  
  FETCH c1 INTO v_apellido;  
  WHILE c1%FOUND LOOP  
    DBMS_OUTPUT.PUT_LINE(v_apellido);  
    FETCH c1 INTO v_apellido;  
  END LOOP;  
  CLOSE c1;  
END;
```

El programa sustituirá la variable por su valor en el momento en que se abre el cursor, y se seleccionarán las filas según dicho valor. Aunque ese valor cambie durante la recuperación de los datos con FETCH, el conjunto de filas que contiene el cursor no variará.

También podíamos haber usado directamente el parámetro formal *dep* en lugar de la variable *v_dept*. El resultado es el mismo.

Una vez creado el procedimiento, se puede ejecutar:

```
SQL> EXECUTE ver_emple_por_dept(30);  
ARROYO  
SALA  
MARTIN  
NEGRO  
TOVAR  
JIMENO
```

We'll write a PL/SQL block that displays the last name and start date of all employees sorted by start date.

3 Escribiremos un bloque PL/SQL que visualice el apellido y la fecha de alta de todos los empleados ordenados por fecha de alta.

1. Mediante una estructura cursor FOR...LOOP.

```
DECLARE
  CURSOR c_emple IS
    SELECT apellido, fecha_alt FROM emple
    ORDER BY fecha_alt;
BEGIN
  FOR v_reg_emp IN c_emple LOOP
    DBMS_OUTPUT.PUT_LINE(v_reg_emp.apellido||'*'||
      v_reg_emp.fecha_alt);
  END LOOP;
END;
```

2. Utilizando un bucle WHILE.

```
DECLARE
  CURSOR c_emple IS
    SELECT apellido, fecha_alt FROM emple
    ORDER BY fecha_alt;
  v_reg_emp c_emple%ROWTYPE;
BEGIN
  OPEN c_emple;
  FETCH c_emple INTO v_reg_emp;
  WHILE c_emple%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE(v_reg_emp.apellido||'*'||
      v_reg_emp.fecha_alt);
    FETCH c_emple INTO v_reg_emp;
  END LOOP;
  CLOSE c_emple;
END;
```

Podemos observar que el bucle FOR...LOOP realiza implícitamente la mayoría de las operaciones con el cursor (abrir, comprobar, recuperar fila y cerrar).

Write a program that displays, in a format similar to the control or sequence breaks seen in SQL, the following data.

- For each employee: last name and salary.
- For each department: number of employees and sum of department salaries.
- At the end of the list: total number of employees and sum of all salaries.

4 Escribe un programa que muestre, en formato similar a las rupturas de control o secuencia vistas en SQL*Plus lo siguientes datos:

- Para cada empleado: apellido y salario.
- Para cada departamento: número de empleados y suma de los salarios del departamento.
- Al final del listado: número total de empleados y suma de todos los salarios.

```
CREATE OR REPLACE PROCEDURE listar_emple
AS
  CURSOR c1 IS
    SELECT apellido, salario, dept_no FROM emple
  ORDER BY dept_no, apellido;
  vr_emp c1%ROWTYPE;
  dep_ant EMPLE.DEPT_NO%TYPE DEFAULT 0;
  cont_emple NUMBER(4) DEFAULT 0;
  sum_sal NUMBER(9,2) DEFAULT 0;
  tot_emple NUMBER(4) DEFAULT 0;
  tot_sal NUMBER(10,2) DEFAULT 0;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO vr_emp;

    /* Si es el primer Fetch inicializamos dep_ant */
    IF c1%ROWCOUNT = 1 THEN
      dep_ant := vr_emp.dept_no;
    END IF;

    /* Comprobación nuevo departamento (o finalización) y resumen del anterior e ini-
    cialización de contadores y acumuladores parciales */
    IF dep_ant <> vr_emp.dept_no OR c1%NOTFOUND THEN
      DBMS_OUTPUT.PUT_LINE('*** DEPTO: ' || dep_ant ||
        ' NUM. EMPLEADOS: ' || cont_emple ||
        ' SUM. SALARIOS: ' || sum_sal);
      dep_ant := vr_emp.dept_no;
      tot_emple := tot_emple + cont_emple;
```

(Continuación)

```
        tot_sal := tot_sal + sum_sal;
        cont_emple := 0;
        sum_sal := 0;
    END IF;
    EXIT WHEN cl%NOTFOUND; /* Condición de salida del bucle */

/* Escribir Líneas de detalle incrementar y acumular */
    DBMS_OUTPUT.PUT_LINE(RPAD(vr_emp.apellido,10) || ' * '
        ||LPAD(TO_CHAR(vr_emp.salario,'999,999'),12));
    cont_emple := cont_emple + 1;
    sum_sal := sum_sal + vr_emp.salario;

END LOOP;
CLOSE cl;

/* Escribir totales informe */
    DBMS_OUTPUT.PUT_LINE(' ***** NUMERO TOTAL EMPLEADOS: '
        || tot_emple || ' TOTAL SALARIOS: ' || tot_sal);

END listar_emple;
```

The following example receives an employee number and an amount that will be incremented to the corresponding employee's salary. We will use two exceptions, one defined by the user null_salary and the other predefined NO_DATA_FOUND.

- 5 El siguiente ejemplo recibe un número de empleado y una cantidad que se incrementará al salario del empleado correspondiente. Utilizaremos dos excepciones, una definida por el usuario `salario_nulo` y la otra predefinida `NO_DATA_FOUND`.

```
CREATE OR REPLACE
PROCEDURE subir_salario(
    num_empleado INTEGER,
    incremento REAL)
IS
    salario_actual REAL;
    salario_nulo EXCEPTION;

BEGIN
    SELECT salario INTO salario_actual FROM emple
        WHERE emp_no = num_empleado;

    IF salario_actual IS NULL THEN
        RAISE salario_nulo; -- levanta salario_nulo
    END IF;

    UPDATE emple SET salario = salario + incremento
        WHERE emp_no = num_empleado;
```

(Continuación)

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE(num_empleado||'*Err.No encontrado');
    WHEN salario_nulo THEN
        DBMS_OUTPUT.PUT_LINE(num_empleado||'*Err. Salario nulo');
END subir_salario;
```

The following example illustrates what we have studied so far regarding exception handling. We will create a block that defines the `blank_err` exception associated with a programmer-defined error and the `no_space_exception` associated with Oracle error number -1547.

- 6 El siguiente ejemplo ilustra lo estudiado hasta ahora respecto a la gestión de excepciones. Crearemos un bloque donde se define la excepción `err_blanco`s asociada con un error definido por el programador y la excepción `no_hay_espacio` asociándola con el error número -1547 de Oracle.

```
DECLARE
    cod_err number(6);
    vnif varchar2(10);
    vnom varchar2(15);
    err_blanco EXCEPTION;
    no_hay_espacio EXCEPTION;
    PRAGMA EXCEPTION_INIT(no_hay_espacio, -1547);
BEGIN
    SELECT col1, col2 INTO vnif, vnom FROM TEMP2;
    IF SUBSTR(vnom,1,1) <= ' ' THEN
        RAISE err_blanco;

    END IF;
    UPDATE clientes SET nombre = vnom WHERE nif = vnif;
EXCEPTION
    WHEN err_blanco THEN
        INSERT INTO temp2(col1) VALUES ('ERR blancos');
    WHEN no_hay_espacio THEN
        INSERT INTO temp2(col1) VALUES ('ERR tablespace');
    WHEN NO_DATA_FOUND THEN
        INSERT INTO temp2(col1) VALUES ('ERR no habia datos');
    WHEN TOO_MANY_ROWS THEN
        INSERT INTO temp2(col1) VALUES ('ERR demasiados datos');
    WHEN OTHERS THEN
        cod_err := SQLCODE;
        INSERT INTO temp2(col1) VALUES (cod_err);
END;
```

Cabe subrayar respecto a las excepciones que aparecen:

- `err_blanco` hay que declararla y levantarla.
- `no_hay_espacio` hay que declararla y asociarla con el error de Oracle, pero no hay que levantarla.
- `NO_DATA_FOUND` y `TOO_MANY_ROWS` no hay que declararlas ni asociarlas. Tampoco hay que levantarlas.
- El manejador `WHEN OTHERS` cazará cualquier otra excepción e insertará el código de error de Oracle en la tabla `temp2` que suponemos creada.

The following example shows the operation of `RAISE_APPLICATION_ERROR` in a procedure with functionality similar to that studied in case study 7 (`raise_salary`).

- 7 El siguiente ejemplo muestra el funcionamiento de `RAISE_APPLICATION_ERROR` en un procedimiento de funcionalidad similar al estudiado en el caso práctico 7 (*subir_salario*).

```
CREATE OR REPLACE
PROCEDURE subir_sueldo
  (Num_emple NUMBER, incremento NUMBER)
IS
  salario_actual NUMBER;
BEGIN
  SELECT salario INTO salario_actual FROM empleados
    WHERE emp_no = num_emple;
  IF salario_actual IS NULL THEN
    RAISE_APPLICATION_ERROR(-20010, ' Salario Nulo');
  ELSE
    UPDATE empleados SET sueldo = salario_actual +
      incremento WHERE emp_no = num_emple;
  ENDIF
END subir_sueldo;
```

Write a procedure that receives all the data of a new employee and processes the registration transaction, managing possible errors. The procedure must specifically manage the following points.

6 Escribe un procedimiento que reciba todos los datos de un nuevo empleado y procese la transacción de alta, gestionando posibles errores. El procedimiento deberá gestionar en concreto los siguientes puntos:

- no_existe_departamento.
- no_existe_director.
- numero_empleado_duplicado.
- Salario nulo: con RAISE_APPLICATION_ERROR.
- Otros posibles errores de Oracle visualizando código de error y el mensaje de error.

1 Supongamos que disponemos de la siguiente vista:

```
CREATE VIEW EMPLEAD AS
SELECT EMP_NO, APELLIDO, OFICIO, DNOMBRE, LOC
FROM EMPLE, DEPART
WHERE EMPLE.DEPT_NO = DEPART.DEPT_NO;
```

Los usuarios verán los datos:

```
SQL> select * from emplead;
EMP_NO  APELLIDO  OFICIO      DNOMBRE      LOC
-----  -
7839    REY       PRESIDENTE  CONTABILIDAD SEVILLA
7876    ALONSO    EMPLEADO    INVESTIGACIÓN MADRID
7521    SALA      VENDEDOR    VENTAS        BARCELONA
...      ...      ...         ...         ...
```

Las siguientes operaciones de manipulación sobre los datos de la vista darán como resultado:

```
SQL> INSERT INTO EMPLEAD VALUES (7999, 'MARTINEZ', 'VENDEDOR', 'CONTABILIDAD', 'SEVILLA');
ERROR en línea 1: ORA-01776: no se puede modificar más de una tabla base a través de una vista.
```

```
SQL> UPDATE EMPLEAD SET DNOMBRE = 'CONTABILIDAD' WHERE APELLIDO = 'SALA';
ERROR en línea 1: ORA-01779: no se puede modificar una columna que se corresponde con una tabla reservada por clave
```


(Continuación)

Para facilitar estas operaciones de manipulación crearemos el siguiente disparador de sustitución:

```
CREATE OR REPLACE TRIGGER t_ges_emplead
INSTEAD OF DELETE OR INSERT OR UPDATE
ON emplead
FOR EACH ROW
DECLARE
    v_dept depart.dept_no%TYPE;
BEGIN
    IF DELETING THEN                /* Si se pretende borrar una fila */
        DELETE FROM EMPLE WHERE emp_no = :old.emp_no;
    ELSIF INSERTING THEN            /* Si se intenta insertar una fila */
        SELECT dept_no INTO v_dept FROM depart
        WHERE depart.dnombre = :new.dnombre
        AND loc = :new.loc;
        INSERT INTO EMPLE (emp_no, apellido, oficio, dept_no)
        VALUES (:new.emp_no, :new.apellido, :new.oficio, v_dept);
    ELSIF UPDATING('dnombre') THEN  /* Si se trata de actualizar
                                     la columna dnombre*/
        SELECT dept_no INTO v_dept FROM depart
        WHERE dnombre = :new.dnombre;
        UPDATE emple SET dept_no = v_dept
        WHERE emp_no = :old.emp_no;
    ELSIF UPDATING('oficio') THEN    /* Si se pretende actualizar
                                     la columna oficio */
        UPDATE emple SET oficio = :new.oficio
        WHERE emp_no = :old.emp_no;
    ELSE
        RAISE_APPLICATION_ERROR(-20500, 'Error en la actualización');
    END IF;
END;
/
```

Ahora podemos realizar las operaciones anteriormente indicadas. Se puede cambiar a un empleado de departamento indicando el nombre del departamento nuevo. El disparador se encargará de comprobar y asignar el número de departamento que corresponda. También se han limitado las columnas que hay que actualizar. En caso de que la operación que pretende el usuario no se contemple entre las alternativas, el *trigger* levantará un error en la aplicación haciendo que falle toda la actualización.

We will write a trigger that will control user connections to the database.

2 Escribiremos un disparador que controlará las conexiones de los usuarios en la base de datos.

Para ello introducirá en la tabla `control_conexiones` el nombre de usuario (`USER`), la fecha y hora en la que se produce el evento de conexión, y la operación `CONEXIÓN` que realiza el usuario.

```
CREATE OR REPLACE TRIGGER ctrl_conexiones
AFTER LOGON
ON DATABASE
BEGIN
    INSERT INTO control_conexiones (usuario, momento, evento)
    VALUES (ORA_LOGIN_USER, SYSTIMESTAMP, ORA_SYSEVENT);
END;
```

Para que el disparador pueda crearse deberá estar creada la tabla `control_conexiones`:

```
CREATE TABLE control_conexiones (usuario VARCHAR2(20),
    momento TIMESTAMP, evento VARCHAR2(20));
```

Para crear este disparador a nivel `ON DATABASE` hay que tener el privilegio `ADMINISTER DATABASE TRIGGER`, de lo contrario sólo nos permitirá crearlo `ON SCHEMA`.

Una vez creado el disparador cualquier evento de conexión en el esquema producirá el disparo del *trigger* y la consiguiente inserción de la fila en la tabla.

USUARIO	MOMENTO	EVENTO
-----	-----	-----
FERNANDO	29/03/06 10:54:51,145000	LOGON

Por otro parte, crearemos un *trigger* que inserte en la tabla `control_eventos` cualquier instrucción de definición de datos:

```
CREATE TABLE control_eventos (usuario VARCHAR2(20), momento TIMESTAMP, evento
    VARCHAR2(40));
```

```
CREATE OR REPLACE TRIGGER ctrl_eventos
AFTER DDL
ON DATABASE
BEGIN
    INSERT INTO control_eventos (usuario, momento, evento)
    VALUES (USER, SYSTIMESTAMP, ORA_SYSEVENT || '*' || ORA_DICT_OBJ_NAME);
END;
```

We will write PL/SQL which will do the following.

- Declare a cursor a cursor based on a query.
- Define a record type supported by the cursor.
- Define a VARRAT type whose elements are of the previously defined record type.
- Declare initialize and use a variable of type VARRAY loading the content of the cursor in the elements and then displaying the content of these.

3 Escribiremos un bloque PL/SQL que realizará lo siguiente:

- Declarar un cursor basado en una consulta.
- Definir un tipo de registro compatible con el cursor.
- Definir un tipo de VARRAY cuyos elementos son del tipo registro previamente definido.
- Declarar inicializar y usar una variable de tipo VARRAY cargando el contenido del cursor en los elementos y posteriormente mostrando el contenido de estos.

```
DECLARE
/* Declaramos un cursor basado en una consulta */
CURSOR c_depar IS
    SELECT dnombre, count(emp_no) numemple
    FROM depart, emple
    WHERE depart.dept_no = emple.dept_no
    GROUP BY depart.dept_no, dnombre;

/* Definimos un tipo compatible con el cursor */
TYPE tr_depto IS RECORD (
    nombredp depart.dnombre%TYPE,
    numemple INTEGER
);

/* Definimos un tipo VARRAY basado en el tipo anterior */
TYPE tv_depto IS VARRAY (6) OF tr_depto;

/* Declaramos e inicializamos una variable del tipo VARRAY definido arriba */
va_departamentos tv_depto := tv_depto(NULL,NULL,NULL,NULL,NULL,NULL);

/* Declaramos una variable para usarla como índice */
n INTEGER := 0;
BEGIN

    /* Cargar valores en la variable */
    FOR vc IN c_depar LOOP
        n := c_depar%ROWCOUNT;
        va_departamentos(n) := vc;
    END LOOP;

    /* Mostrar los datos de la variable */
    FOR i IN 1..n LOOP
        DBMS_OUTPUT.PUT_LINE(' * Dnombre: ' || va_departamentos(i).nombredp ||
                               ' * N°Empleados: ' || va_departamentos(i).numemple );
    END LOOP;
END;
```

Next we will rewrite the code from Case Study 3 using an indexed table and attributes to traverse the table.

- 4 A continuación reescribiremos el código del Caso práctico 3 usando una tabla indexada y los atributos de la tabla para recorrer la tabla.

```
DECLARE
    CURSOR c_depar IS          /* Declaramos un cursor basado en una consulta */
        SELECT depart.dept_no, dnombre, count(emp_no) numemple
        FROM depart, emple
        WHERE depart.dept_no = emple.dept_no
        GROUP BY depart.dept_no, dnombre;

    TYPE tr_depto IS RECORD (   /* Definimos un tipo compatible con el cursor */
        nombredp depart.dnombre%TYPE,
        numemple INTEGER);

    /* Definimos un tipo TABLA INDEXADA basado en el tipo anterior */
    TYPE ti_depto IS TABLE OF tr_depto INDEX BY PLS_INTEGER;

    va_departamentos ti_depto; /* Declaramos la variable del tipo TABLA INDEXADA */

    n PLS_INTEGER := 0;        /* Declaramos una variable para usarla como índice */
BEGIN
    FOR vc IN c_depar LOOP      /* Cargar valores. El índice es el N° de departamento */
        va_departamentos(vc.dept_no).nombredp := vc.dnombre;
        va_departamentos(vc.dept_no).numemple := vc.numemple;
    END LOOP;

    n := va_departamentos.FIRST;
    WHILE va_departamentos.EXISTS(n) LOOP /* Mostrar los datos de la variable */
        DBMS_OUTPUT.PUT_LINE(' * Dep N° : ' || n ||
                               ' * Dnombre: ' || va_departamentos(n).nombredp ||
                               ' * N° Empleados: ' || va_departamentos(n).numemple);
        n := va_departamentos.NEXT(n);
    END LOOP;
END;
```

TEST

Las variables son: **Se utiliza y se les asignará nuevos valores en el ejecutable**

Las Variables pueden ser utilizadas para: La manipulación de los valores almacenados.

Ventajas de PL/SQL **Integración de procedimiento construye con SQL**

Unidades Léxicas Son bloques de construcción de cualquier bloque PL/SQL

Directriz Lateral de PL/SQL Los números pueden ser valores simples o científica

PL/SQL Proporciona una estructura de bloques para las unidades de ejecutables

¿Con qué procedimiento puedo realizar una Salida de un bloque PL/SQL?
DBMS_OUTPUT.PUT_LINE

¿Qué es PL/SQL? **Procedural Language for Structured Query Language**

¿Con qué variable puedo hacer uso de declaración? DECLARE

Los identificadores se utilizan para: Proporcionar una convención para nombres de variables

Test 2

La forma de abandonar cualquier control de flujo etiquetado en un procedimiento almacenado es con: Leave.

Cambiar el delimitador por otro valor del ";" antes de la creación de un procedimiento almacenado sirve para pasar el delimitador ; usado en el cuerpo del procedimiento a través del servidor en lugar de ser interpretado por el mismo . Verdadero

Los cursores deben declararse después de los handlers: Falso

Los cursores nos permiten almacenar un conjunto de filas de una tabla en una estructura de datos que podemos ir recorriendo de forma secuencial. Verdadero

La sentencia SELECT id,data INTO x,y FROM test.t1 LIMIT 1; : Introduce en las variables x e y los valores de los atributos id y data resultado de la consulta.

Seleccione la/s afirmación/es correcta/s: La cláusula RETURNS puede especificarse sólo con FUNCTION , donde es obligatorio.

Las sentencias que pueden activar el disparador (TRIGGER) son: INSERT, UPDATE y DELETE.

La operación que podemos utilizar para ir obteniendo las filas de un cursor abierto es: FETCH

Selecciona la afirmación correcta: d.Todas las anteriores.

El comando para ver la definición de un procedimiento almacenado es: **SHOW CREATE PROCEDURE <nombre_procedimiento>**

Para que un handler continúe la rutina actual tras la ejecución del comando handler, el tipo en la declaración debería ser: **CONTINUE**

El uso de SET NEW.nombre_col = valor y SET nombre_var = NEW.nombre_col necesita que se tengan los siguientes privilegios respectivamente: **UPDATE y SELECT sobre la columna.**

Los procedimientos almacenados pueden utilizar LOAD DATA INFILE. Falso

La sintaxis para declarar un cursor es la siguiente: DECLARE nombre_cursor CURSOR FOR sentencia_select. Indica las afirmaciones falsas: **El comando SELECT puede tener la cláusula INTO.**

El disparador (TRIGGER) puede activarse antes (BEFORE) ó después (AFTER) de la sentencia que lo activa. **Verdadero**

Las columnas de la tabla asociada con el disparador pueden referenciarse empleando los alias OLD y NEW. Para hacer referencia a una columna de una fila existente, antes de ser actualizada o borrada utilizaríamos: **OLD.nombre_col**

Para el manejo de errores se utiliza: **Handlers**

Un disparador (TRIGGER) es un objeto con nombre en una base de datos que se asocia con una tabla, y se activa cuando ocurre un evento en particular para esa tabla. **Verdadero**

Indica la afirmación correcta. Los constructores de control de flujo son: **IF, CASE, LOOP, WHILE, ITERATE y LEAVE. Los bucles FOR no están soportados.**

Las secuencias de operaciones a utilizar con cursores son: **DECLARE, OPEN, FETCH, CLOSE**

TEST 3

Can you commit inside a Trigger? Yes, in autonomous transactions

Maximum characters allowed in dbms_out.put_line() 255 chars

Can you delete a column in table with data in Oracle? Yes, always

Maximum number of columns in a table or view in Oracle 9i? 1000

What is the Datatype of NULL in Oracle? Char(0)

How to check the version of Oracle? Select * from v\$version;

Maximum levels of subqueries in the WHERE clause of an SQL statement? 255

You need to calculate the total of all salaries in the accounting department. Which group function should you use? SUM

Which of the following functions are available in SQL? TRUNCATE.

Which one is a system privilege? CREATE TABLE

Which of the following statements contains an error? SELECT empid WHERE empid = 56949 AND lastname = 'SMITH';

The command to remove rows from a table 'CUSTOMER' is: DELETE FROM CUSTOMER WHERE...

Which is an /SQL*Plus command? DESCRIBE

Which one of the following sorts rows in SQL? ORDER BY

Which SQL statement is used to insert new data in a database? INSERT INTO

EJERCICIO RARO

Count Users

Consider the following code:

```

CREATE OR REPLACE FUNCTION count_users
RETURN NUMBER
IS
    user_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO user_count FROM users;
    RETURN user_count;
END;
/

```

Select the statements that are correct.

(Select all acceptable answers.)

When the *users* table is dropped, the *count_users* function is invalidated.

When the *users* table is dropped any function, procedure, or package using *count_users* is invalidated.

6. Mostrar los numeros del 1 al 100 con un loop.

```

declare
    i number(8) := 1;
begin

    loop
        DBMS_OUTPUT.PUT_LINE(i);
        exit when i=10;
        i := i+1;
    end loop;

end;
/

```

Mostrar los numeros del 1 al 100 con un while.

```

declare
    i number(8) := 1;
begin

```

```

while (i<=10)
loop
    DBMS_OUTPUT.PUT_LINE(i);
    i := i+1;
end loop;

```

end;

/

Mostrar los numeros del 1 al 100 con un for.

begin

```

for i in 1..10
loop
    DBMS_OUTPUT.PUT_LINE(i);
end loop;

```

end;

/

-- De 10 a 1

begin

```

for i in reverse 1..10
loop
    DBMS_OUTPUT.PUT_LINE(i);
end loop;

```

end;

/

13. Realizar una función que me devuelva la suma de pagos que ha realizado.
Pasa el código por parametro. Controla en caso de que no se encuentre, en ese caso devuelve un -1.


```

create or replace function Pagos_cliente(v_codigocliente
clientes.codigocliente%type)
return Number
as
    v_sumapagos pagos.cantidad%type := 0;
begin

    select sum(cantidad) into v_sumapagos
    from pagos
    where codigocliente = v_codigocliente;

    if v_sumapagos is null then
        raise no_data_found;
    else
        return v_sumapagos;
    end if;
exception
    when no_data_found then
        return -1;

end;
/

declare
    v_codigocliente clientes.codigocliente%type := &codigo;
    v_suma pagos.cantidad%type;
begin
    v_suma := Pagos_cliente(v_codigocliente);
    if v_suma = -1 then
        DBMS_OUTPUT.PUT_LINE('El cliente no existe');
    else
        DBMS_OUTPUT.PUT_LINE('La suma de pagos es ' || v_suma);
    end if;
end;

```

```
end if;
```

```
end;
```

```
/
```

18. Crear un cursor para ver todos los clientes que no hayan hecho pagos. Hazlo con un loop.

```
declare
```

```
    v_nombrecliente clientes.nombrecliente%type;
```

```
    cursor clientes_sin_pagos_cursor is
```

```
        select nombrecliente
```

```
        from clientes c
```

```
        where not exists(select codigocliente from pagos where codigocliente =  
c.codigocliente);
```

```
begin
```

```
    open clientes_sin_pagos_cursor;
```

```
    loop
```

```
        fetch clientes_sin_pagos_cursor into v_nombrecliente;
```

```
        exit when clientes_sin_pagos_cursor%notfound;
```

```
        dbms_output.put_line(v_nombrecliente);
```

```
    end loop;
```

```
    close clientes_sin_pagos_cursor;
```

```
end;
```

```
/
```

19. Crear un cursor para ver todos los clientes que no hayan hecho pagos. Hazlo con un for.

```
declare
```

```

cursor clientes_sin_pagos_cursor is
    select nombrecliente
    from clientes c
    where not exists(select codigocliente from pagos where codigocliente =
c.codigocliente);
begin

    for registro in clientes_sin_pagos_cursor loop

        dbms_output.put_line(registro.nombrecliente);

    end loop;

end;
/

```

1.-What command extracts data from the database?

3.33 SELECT

2.- What command updates data in the database?

3.33 UPDATE

3.- Command that allows modifying the structure of an object.

3.33 ALTER

4.- Command that deletes an object from the database.

3.33 DROP

5.- SQL statement that adds one or more records to a table.

0 CREATE

3.33 INSERT

6.- An SQL statement that is used to modify the values in a table.

3.33 UPDATE

7.- Command that creates an object within the database.

3.33 CREATET

8.- SQL statement that allows you to delete zero or more records in a table.

0 DROP

3.33 DELETE

9.-What would be the correct syntax to select the column called "Surnames" from a table called "Workers"?

0 EXTRACT Surname FROM Workers

☐ SELECT Workers, Surnames

3.33 SELECT Surname FROM Workers

10.-What is the correct syntax to select all the fields of a table called "Workers"?

3.33 SELECT * FROM Workers

☐ SELECT [all] FROM Workers

☐ SELECT Workers

11- What is the correct syntax to select all the fields of a table called Workers, in which the "Name" column is "Raúl"?

☐ SELECT * FROM Workers WHERE Name:'Raul'

3.33 SELECT * FROM Workers WHERE Name='Raúl'

☐ SELECT * FROM Workers LIKE 'Raul'

12.-What keyword is used to return only different values?

☐ NOTICE

☐ COUNT

3.33 DISTINCT

13.- What keyword is used to order the result?

☐ ORDER

☐ SORT

3.33 ORDER BY

14.- What is the correct syntax if you want to insert a new row in the "Workers" table (whose fields are "Name" and "Surname")?

☐ INSERT ('Carlos', 'Perez') INTO Workers

3.33 INSERT INTO Workers VALUES ('Carlos', 'Perez')

☐ INSERT VALUES ('Carlos', 'Perez') INTO Workers

15.- How would you change "Carlos" to "Javier" in the "Name" column of the "Workers" table?

☐ UPDATE Workers SET Name = 'Carlos' INTO Name = 'Javier'

3.33 UPDATE Workers SET Name = 'Javier' WHERE Name = 'Carlos'

☐ SAVE Workers SET Name = 'Carlos' INTO Name = 'Javier'

16.- What is the correct syntax to delete the records whose "Name" field is "Andrés"?

3.33 DELETE FROM Workers WHERE Name = 'Andres'

☐ DELETE ROW Name='Andrew' FROM Workers

☐ DELETE Name='Andrew' FROM Workers

17.- If we wanted to count the number of records in the "Workers" table...

☐ SELECT NUMBER FROM Workers

☐ SELECT COUNT Workers

3.33 SELECT COUNT(*) FROM Workers

18.--You could use: [DELETE FROM user] , to delete the user table

3.33 false

0 true

19.- To eliminate all the rows of a table called "AUTHOR" you could use the sentence [TRUNCATE TABLE autor].

0 false

3.33 true

20.--To show all the records of the table called "nationality" you could apply:

SELECT FROM * nationality

3.33 false

0 true

21.- SELECT MAX(idusuario) FROM user , it would allow me to see the first record of the user table

3.33 false

0 true

22.- SELECT TOP 1 userid from user ORDER BY userid DESC, it allows me to show the first record of the table

3.33 false

0 true

23.-- SQL statement to filter all users named "JUAN"; SELECT * FROM USER WHERE name like '%JOHN%'

0 false

3.33 true

24.- Show all the fields and all the records of users whose userid is 'LEON'; [SELECT username, password, name, phone FROM username WHERE username='LEON' AND password='123']

3.33 false

0 true

25.- Show all the fields and all the records of users whose password is "123"; [SELECT * FROM user WHERE password='123']

0 false

3.33 true

26.--Show how many records of the table called employee, have the name ESMERALDA;

[SELECT COUNT(name) FROM employee WHERE name='ESMERALDA']

0 false

3.33 true

27.-Insert a record in the loan table:

INSERT INTO loan(loanid, date, userid, idlector, bookid, typeid) VALUES
(7,CURRENT_TIMESTAMP,9,8,11,2)

3.33 false

0 true

28.-Insert a record in the loan table:

INSERT INTO loan(date, userid, idlector, bookid,typeid) VALUES
(CURRENT_TIMESTAMP,9,'UBUNTU',11,2)

0 false

3.33 true

29.-To Delete all the rows of a table called "AUTHOR" you could use the Sentence:

[DELETE FROM author].

0 false

3.33 true

30 What does SQL stand for?

3.33 Structured Query Language

0 Structured Question Line

0 Strong Question Language