**Desarrollo de aplicaciones multiplataforma**

**Acceso a datos**

**Ejercicios obligatorios-Tema 1**

**Mohamed El Younousi Bentalha**

1. Realizar un programa que solicite un número de empleado determinado, el programa debe comprobar si el salario es mayor que 1000, y si no es así actualizar el salario con una subida del 20% y, en una tabla temporal, insertar el nombre del empleado y el mensaje "Sueldo actualizado". Si ya superaba los 1000, se inserta en la tabla temporal el nombre del empleado y el mensaje "No necesita actualización".

**RESPUESTA:**

drop table temp;

create table temp (codempl NUMBER(4), mensaje VARCHAR2(50));

set serveroutput on;

set VERIFY off;

DECLARE

v\_cdEmpleado employee.employee\_id%TYPE;

v\_salario employee.salary %TYPE;

BEGIN

select employee\_id, salary into v\_cdEmpleado, v\_salario from employee where employee\_id = &v\_cdEmpleado;

if(v\_salario < 1000) then

v\_salario := v\_salario+(v\_salario \* 20 /100);

update employee set salary = v\_salario where employee.employee\_id=v\_cdEmpleado;

insert into temp values(v\_cdEmpleado, 'Salario actualizado.');

else

insert into temp values(v\_cdEmpleado, 'No necesita actualizacion.');

END IF;

END;

/

1. Hacer un programa que solicite por teclado un código de cliente e inserte en la tabla temporal el número total de pedidos de ese cliente (tot\_ped), el precio total de esos pedidos (precio\_tot), el código del cliente (cod\_cli) y el nombre del cliente (nombre\_cli).

**RESPUESTA:**

drop table temp2;

create table temp2 (cod\_cli NUMBER(4), nombre\_cli VARCHAR2(50), tot\_ped NUMBER(4), precio\_tot NUMBER(6));

set serveroutput on;

set VERIFY off;

DECLARE

v\_customer\_id NUMBER(6);

BEGIN

dbms\_output.put\_line('Ingrese el código de cliente: ');

v\_customer\_id := TO\_NUMBER(TO\_CHAR(&v\_customer\_id));

insert into temp2 (cod\_cli, nombre\_cli, tot\_ped, precio\_tot)

select c.customer\_id, c.name, count(so.order\_id) as tot\_ped, sum(so.total) as precio\_tot

from customer c

left join sales\_order so on c.customer\_id = so.customer\_id

where c.customer\_id = v\_customer\_id

group by c.customer\_id, c.name;

END;

/

1. Crearse un registro t\_reg\_emple con la siguiente estructura: código del empleado, nombre y el job. Insertar en una tabla temporal el nombre del empleado y job para el empleado 7782.

**RESPUESTA:**

create table t\_reg\_emple (

emp\_id NUMBER(4, 0),

emp\_name VARCHAR2(30),

emp\_job VARCHAR2(30)

);

set serveroutput on;

set VERIFY off;

DECLARE

v\_emp\_id NUMBER(4) := 7782;

v\_emp\_name VARCHAR2(30);

v\_emp\_job VARCHAR2(30);

BEGIN

select e.last\_name || ', ' || e.first\_name, j."function"

into v\_emp\_name, v\_emp\_job

from "emple"."employee" e

join "emple"."job" j on e.job\_id = j.job\_id

where e.employee\_id = v\_emp\_id;

insert into t\_reg\_emple (emp\_id, emp\_name, emp\_job)

values (v\_emp\_id, v\_emp\_name, v\_emp\_job);

END;

/

1. Insertar 50 filas en una tabla. Recorre todas las filas de la tabla y las inserta en una tabla temporal. Crear la tabla students e insertar algunos datos.

**RESPUESTA:**

create table temp\_results (

id NUMBER,

descrp VARCHAR2(50)

);

create table prueba1 (

id NUMBER,

descrp VARCHAR2(50)

);

set serveroutput on;

set VERIFY off;

DECLARE

v\_clm1 NUMBER;

v\_clm2 VARCHAR2(50);

BEGIN

FOR i IN 1..50 LOOP

INSERT INTO prueba1 (id, descrp) VALUES (i, 'Valor ' || i);

END LOOP;

FOR rec IN (SELECT \* FROM prueba1) LOOP

v\_clm1 := rec.id;

v\_clm2 := rec.descrp;

INSERT INTO temp\_results (id, descrp) VALUES (v\_clm1, v\_clm2);

END LOOP;

END;

/

CREATE TABLE students (

id NUMBER,

nombre VARCHAR2(50)

);

INSERT INTO students (id, nombre) VALUES (1, 'Pedro');

INSERT INTO students (id, nombre) VALUES (2, 'Raul');

INSERT INTO students (id, nombre) VALUES (3, 'Rosa');

INSERT INTO students (id, nombre) VALUES (4, 'Maria');

1. Llenar una tabla PL con los nombres de los clientes y cargarla en una tabla temporal de sql que contenga el índice de la tabla PL y el nombre del cliente, y visualizar ésta.

**RESPUESTA:**

CREATE GLOBAL TEMPORARY TABLE temp\_customer\_data (

index\_number NUMBER,

customer\_name VARCHAR2(100)

) ON COMMIT DELETE ROWS;

DECLARE

TYPE customer\_table\_type IS TABLE OF VARCHAR2(100);

customer\_table customer\_table\_type := customer\_table\_type();

v\_index NUMBER;

v\_customer\_name VARCHAR2(100);

BEGIN

SELECT NAME

BULK COLLECT INTO customer\_table

FROM "EMPLE"."CUSTOMER";

FOR i IN 1..customer\_table.COUNT LOOP

v\_index := i;

v\_customer\_name := customer\_table(i);

INSERT INTO temp\_customer\_data (index\_number, customer\_name)

VALUES (v\_index, v\_customer\_name);

END LOOP;

FOR rec IN (SELECT \* FROM temp\_customer\_data) LOOP

DBMS\_OUTPUT.PUT\_LINE('Índice: ' || rec.index\_number || ', Nombre del Cliente: ' || rec.customer\_name);

END LOOP;

END;

/

1. Llenar una tabla temporal con los códigos de clientes, nombre del cliente y código de empleado de tres formas posibles
   1. **usando bucle LOOP,**

CREATE GLOBAL TEMPORARY TABLE temp\_ej6LOOP (

codigo\_cliente NUMBER,

nombre\_cliente VARCHAR2(100),

codigo\_empleado NUMBER

) ON COMMIT DELETE ROWS;

DECLARE

v\_codigo\_cliente customer.customer\_id%TYPE;

v\_nombre\_cliente customer.name%TYPE;

v\_codigo\_empleado employee.employee\_id%TYPE;

v\_counter NUMBER := 1;

BEGIN

LOOP

SELECT c.customer\_id, c.name, e.employee\_id

INTO v\_codigo\_cliente, v\_nombre\_cliente, v\_codigo\_empleado

FROM customer c, employee e

WHERE ROWNUM = v\_counter;

EXIT WHEN v\_counter > 10;

INSERT INTO temp\_ej6LOOP VALUES (v\_codigo\_cliente, v\_nombre\_cliente, v\_codigo\_empleado);

v\_counter := v\_counter + 1;

END LOOP;

COMMIT;

END;

/

* 1. **usando bucle WHILE**

CREATE GLOBAL TEMPORARY TABLE temp\_ej6WHILE (

codigo\_cliente NUMBER,

nombre\_cliente VARCHAR2(100),

codigo\_empleado NUMBER

) ON COMMIT DELETE ROWS;

DECLARE

v\_codigo\_cliente customer.customer\_id%TYPE;

v\_nombre\_cliente customer.name%TYPE;

v\_codigo\_empleado employee.employee\_id%TYPE;

v\_counter NUMBER := 1;

BEGIN

WHILE v\_counter <= 10 LOOP

SELECT c.customer\_id, c.name, e.employee\_id

INTO v\_codigo\_cliente, v\_nombre\_cliente, v\_codigo\_empleado

FROM customer c, employee e

WHERE ROWNUM = v\_counter;

INSERT INTO temp\_ej6WHILE VALUES (v\_codigo\_cliente, v\_nombre\_cliente, v\_codigo\_empleado);

v\_counter := v\_counter + 1;

END LOOP;

COMMIT;

END;

/

* 1. **usando bucle FOR.**

CREATE GLOBAL TEMPORARY TABLE temp\_ej6FOR (

codigo\_cliente NUMBER,

nombre\_cliente VARCHAR2(100),

codigo\_empleado NUMBER

) ON COMMIT DELETE ROWS;

DECLARE

v\_codigo\_cliente customer.customer\_id%TYPE;

v\_nombre\_cliente customer.name%TYPE;

v\_codigo\_empleado employee.employee\_id%TYPE;

BEGIN

FOR i IN 1..10 LOOP

SELECT c.customer\_id, c.name, e.employee\_id

INTO v\_codigo\_cliente, v\_nombre\_cliente, v\_codigo\_empleado

FROM customer c, employee e

WHERE ROWNUM = i;

INSERT INTO temp\_ej6FOR VALUES (v\_codigo\_cliente, v\_nombre\_cliente, v\_codigo\_empleado);

END LOOP;

COMMIT;

END;

/

1. A los pedidos del cliente 2, añadir 10 unidades más a la cantidad e insertar en una tabla temporal el código de artículo y un mensaje que diga "Diez unidades más vendidas".

**RESPUESTA:**

-- Crear una tabla temporal

CREATE GLOBAL TEMPORARY TABLE TEMP\_ORDER\_UPDATE (

PRODUCT\_ID NUMBER(6,0),

MESSAGE VARCHAR2(50)

) ON COMMIT PRESERVE ROWS;

-- Incrementar en 10 unidades la cantidad de los artículos para pedidos del cliente 2

BEGIN

FOR item\_row IN (SELECT I.PRODUCT\_ID, I.ORDER\_ID

FROM ITEM I

JOIN SALES\_ORDER S ON I.ORDER\_ID = S.ORDER\_ID

WHERE S.CUSTOMER\_ID = 2)

LOOP

UPDATE ITEM

SET QUANTITY = QUANTITY + 10

WHERE PRODUCT\_ID = item\_row.PRODUCT\_ID AND ORDER\_ID = item\_row.ORDER\_ID;

-- Insertar en la tabla temporal

INSERT INTO TEMP\_ORDER\_UPDATE (PRODUCT\_ID, MESSAGE)

VALUES (item\_row.PRODUCT\_ID, 'Diez unidades más vendidas');

END LOOP;

COMMIT;

END;

1. Crear un procedimiento que añada nuevos pedidos a un cliente determinado. El procedimiento recibe el idped, idart, cantidad y codcli.

**RESPUESTA:**

-- Crear el procedimiento

CREATE OR REPLACE PROCEDURE ADD\_NEW\_ORDER(

p\_idped NUMBER,

p\_idart NUMBER,

p\_cantidad NUMBER,

p\_codcli NUMBER

)

AS

BEGIN

-- Insertar un nuevo pedido

INSERT INTO SALES\_ORDER (ORDER\_ID, ORDER\_DATE, CUSTOMER\_ID, TOTAL)

VALUES (p\_idped, SYSDATE, p\_codcli, 0);

-- Insertar el artículo en la tabla ITEM

INSERT INTO ITEM (ORDER\_ID, PRODUCT\_ID, QUANTITY)

VALUES (p\_idped, p\_idart, p\_cantidad);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Nuevo pedido agregado exitosamente.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

1. Procedimiento que borra los pedidos del cliente especificado. Recibe el codcli.

**RESPUESTA:**

-- Crear el procedimiento

CREATE OR REPLACE PROCEDURE DELETE\_ORDERS\_FOR\_CUSTOMER(

p\_codcli NUMBER

)

AS

BEGIN

-- Borrar los pedidos del cliente

DELETE FROM SALES\_ORDER WHERE CUSTOMER\_ID = p\_codcli;

DBMS\_OUTPUT.PUT\_LINE('Pedidos del cliente ' || p\_codcli || ' borrados exitosamente.');

END;

/

1. Procedimiento para cambiar el oficio de un empleado. Se pasa al codemp y el nuevo oficio. Insertará en TEMP oficio\_ant, nuevo, codemp.

**RESPUESTA:**

CREATE GLOBAL TEMPORARY TABLE TEMP\_EJ10 (

COD\_EMP NUMBER,

OFICIO\_ANTERIOR NUMBER,

NUEVO\_OFICIO NUMBER

) ON COMMIT DELETE ROWS;

CREATE OR REPLACE PROCEDURE CHANGE\_EMPLOYEE\_JOB(

p\_codemp NUMBER,

p\_new\_job NUMBER

)

AS

v\_old\_job NUMBER;

BEGIN

-- Verificar si el empleado existe y obtener su oficio actual

SELECT JOB\_ID INTO v\_old\_job FROM EMPLOYEE WHERE EMPLOYEE\_ID = p\_codemp;

-- Actualizar el oficio del empleado

UPDATE EMPLOYEE

SET JOB\_ID = p\_new\_job

WHERE EMPLOYEE\_ID = p\_codemp;

-- Insertar el registro en la tabla temporal

INSERT INTO TEMP\_EJ10 (cod\_emp, oficio\_anterior, nuevo\_oficio)

VALUES ( p\_codemp ,v\_old\_job ,p\_new\_job);

DBMS\_OUTPUT.PUT\_LINE('Oficio del empleado ' || p\_codemp || ' cambiado exitosamente.');

END;

/

1. Crear una función "Anual" para devolver el salario anual cuando se pasa el salario mensual y la comisión de un empleado. Asegurarse que controla nulos. Utilizar una variable de acoplamiento para ver lo que devuelve.

**RESPUESTA:**

-- Crear la función "Anual"

CREATE OR REPLACE FUNCTION Anual(

p\_salario\_mensual NUMBER,

p\_comision NUMBER

) RETURN NUMBER

IS

v\_salario\_anual NUMBER;

BEGIN

-- Controlar valores nulos

IF p\_salario\_mensual IS NULL THEN

RETURN NULL;

END IF;

-- Calcular el salario anual

v\_salario\_anual := p\_salario\_mensual \* 12;

-- Añadir la comisión al salario anual

IF p\_comision IS NOT NULL THEN

v\_salario\_anual := v\_salario\_anual + p\_comision;

END IF;

-- Devolver el salario anual

RETURN v\_salario\_anual;

END;

/

SET SERVEROUTPUT ON;

DECLARE

v\_salario\_mensual NUMBER := 2500;

v\_comision NUMBER := 300;

v\_salario\_anual\_resultado NUMBER;

BEGIN

-- Llamar a la función "Anual" con valores de muestra

v\_salario\_anual\_resultado := Anual(v\_salario\_mensual, v\_comision);

-- Mostrar el resultado

DBMS\_OUTPUT.PUT\_LINE('El salario anual es: ' || v\_salario\_anual\_resultado);

END;

/

1. Crear un paquete "Actualiza" que tiene tres procedimientos y una tabla PL:
   1. Un procedimiento de alta de un pedido.
   2. Un procedimiento de baja de los pedidos y detalles de un cliente concreto.
   3. Una tabla PL que almacenará los códigos de los pedidos.
   4. Un procedimiento "Listar" que tiene como parámetro 'in' el código de cliente y como parámetro de salida una tabla PL con los códigos de los pedidos del cliente especificado, además se grabará en una tabla temporal el código del pedido y el código del artículo.

**RESPUESTA:**

CREATE OR REPLACE PACKAGE Actualiza IS

-- Procedimiento de alta de un pedido

PROCEDURE AltaPedido(

p\_codigo\_pedido NUMBER,

p\_codigo\_cliente NUMBER,

p\_fecha\_pedido DATE

);

-- Procedimiento de baja de los pedidos y detalles de un cliente concreto

PROCEDURE BajaPedidosCliente(

p\_codigo\_cliente NUMBER

);

-- Procedimiento "Listar" con parámetro de entrada y de salida

PROCEDURE Listar(

p\_codigo\_cliente IN NUMBER,

p\_tabla\_salida OUT SYS\_REFCURSOR

);

END Actualiza;

/

CREATE OR REPLACE PACKAGE BODY Actualiza IS

-- Tabla PL para almacenar los códigos de los pedidos

TYPE tabla\_pedidos IS TABLE OF NUMBER;

tabla\_pedidos\_actualizada tabla\_pedidos := tabla\_pedidos();

-- Procedimiento de alta de un pedido

PROCEDURE AltaPedido(

p\_codigo\_pedido NUMBER,

p\_codigo\_cliente NUMBER,

p\_fecha\_pedido DATE

) IS

BEGIN

-- Lógica para la inserción de un nuevo pedido

-- Implementación según tus requisitos

NULL;

END AltaPedido;

-- Procedimiento de baja de los pedidos y detalles de un cliente concreto

PROCEDURE BajaPedidosCliente(

p\_codigo\_cliente NUMBER

) IS

BEGIN

-- Lógica para eliminar los pedidos y detalles de un cliente

-- Implementación según tus requisitos

NULL;

END BajaPedidosCliente;

-- Procedimiento "Listar" con parámetro de entrada y de salida

PROCEDURE Listar(

p\_codigo\_cliente IN NUMBER,

p\_tabla\_salida OUT SYS\_REFCURSOR

) IS

BEGIN

-- Lógica para listar los códigos de los pedidos del cliente y guardarlos en una tabla temporal

-- Implementación según tus requisitos

NULL;

END Listar;

END Actualiza;

/

1. Crear un trigger para ver como suceden los eventos de activación. Vamos a utilizar la tabla employe y vamos a cambiar a los "Manager" por "Analist".

**RESPUESTA:**

CREATE OR REPLACE TRIGGER actualiza\_mang\_analyst

BEFORE UPDATE OF job\_id ON employee

FOR EACH ROW

BEGIN

IF UPDATING THEN

IF :OLD.job\_id = 'Manager' THEN

:NEW.job\_id := 'Analyst';

DBMS\_OUTPUT.PUT\_LINE('Manager se ha actualizado a Analyst');

END IF;

END IF;

END;

/

1. Crear un disparador a nivel de fila tal que después de insertar, modificar o borrar un detalle de la tabla detalles introduzca en la tabla temporal el usuario, la fecha del sistema el código de pedido que se ha modificado, insertado o borrado, así como un mensaje que diga "Detalle dado de alta", "Detalle borrado", "Detalle modificado" según corresponda.

**RESPUESTA:**

CREATE GLOBAL TEMPORARY TABLE tabla\_temporal\_ej14 (

usuario VARCHAR2(100),

fecha DATE,

codigo\_producto NUMBER,

mensaje VARCHAR2(100)

) ON COMMIT DELETE ROWS;

CREATE OR REPLACE TRIGGER product\_change\_trigger

AFTER INSERT OR UPDATE OR DELETE ON product

FOR EACH ROW

DECLARE

info VARCHAR2(100);

BEGIN

IF INSERTING THEN

info := 'Producto dado de alta';

ELSIF UPDATING THEN

info := 'Producto modificado';

ELSIF DELETING THEN

info := 'Producto borrado';

END IF;

INSERT INTO tabla\_temporal\_ej14 (usuario, fecha, codigo\_producto, mensaje)

VALUES (USER, SYSDATE, :NEW.product\_id, info);

END;

/

1. Disparador que inserta una fila en la tabla Temp con el texto ‘subida de salario empleado’ y el número del empleado al que se le ha subido el salario. El disparador se activará después de actualizar el salario de la tabla empleador.

**RESPUESTA:**

CREATE TABLE temp\_ej15 (

texto VARCHAR2(100),

numero\_empleado NUMBER

);

CREATE OR REPLACE TRIGGER tg\_incremento\_salario

AFTER UPDATE OF salary ON employee

FOR EACH ROW

DECLARE

BEGIN

INSERT INTO Temp\_ej15 (texto, numero\_empleado)

VALUES ('Se ha aumentado el salario al empleado', :NEW.employee\_id);

END;

/

1. Trigger que se dispara cada vez que se borra un empleado, guardando el número empleado, apellido y departamento en la tabla TEMP.

**RESPUESTA:**

CREATE TABLE TEMP\_ej16 (

numero\_empleado NUMBER,

apellido VARCHAR2(50),

departamento VARCHAR2(50)

);

CREATE OR REPLACE TRIGGER tg\_empleado\_eliminado

BEFORE DELETE ON employee

FOR EACH ROW

BEGIN

INSERT INTO TEMP\_ej16(numero\_empleado, apellido, departamento)

VALUES (:OLD.employee\_id, :OLD.last\_name, :OLD.department\_id);

END;

/

1. Limitar a 5 (0 al 4) el número de detalles de cada pedido.

**RESPUESTA:**

1. Solicitar un código de cliente por teclado; si existe se inserta su código en una tabla temporal y el mensaje 'EXISTE', y si no existe se dispara la excepción NO\_DATA\_FOUND y se inserta en la tabla temporal el mensaje 'NO EXISTE'.

**RESPUESTA:**

-- Crear la tabla temporal manualmente en la base de datos

CREATE GLOBAL TEMPORARY TABLE temp\_ej18 (

codigo\_cliente NUMBER,

mensaje VARCHAR2(100)

) ON COMMIT DELETE ROWS;

CREATE GLOBAL TEMPORARY TABLE temp\_ej18\_NDF (

mensaje VARCHAR2(100)

) ON COMMIT DELETE ROWS;

-- Luego, ejecutar el bloque PL/SQL sin la parte de creación de tabla

DECLARE

v\_codcli customer.customer\_id%TYPE;

BEGIN

-- Solicitar el código de cliente por teclado

v\_codcli := &v\_codcli; -- &v\_codcli es una entrada del usuario

-- Verificar si el código de cliente existe

SELECT COUNT(\*)

INTO v\_codcli

FROM emple.customer

WHERE CUSTOMER\_ID = v\_codcli;

-- Si existe, realizar la inserción

IF v\_codcli > 0 THEN

INSERT INTO temp\_ej18 (codigo\_cliente, mensaje)

VALUES (v\_codcli, 'EXISTE');

COMMIT;

ELSE

RAISE NO\_DATA\_FOUND;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

INSERT INTO temp\_ej18\_NDF (mensaje)

VALUES ('NO EXISTE');

COMMIT;

END;

/

1. Se solicita un código de artículo por teclado y se inserta en una tabla temporal el precio y la descripción de este artículo si está pedido, es decir, si su código está en la tabla detalle. Si el artículo no está pedido, se genera una excepción de usuario con el mensaje "El artículo (código) no lo ha pedido ningún cliente"

**RESPUESTA:**

DROP TABLE temp\_ej19;

CREATE GLOBAL TEMPORARY TABLE temp\_ej19 (

codigo\_articulo NUMBER,

precio NUMBER,

descripcion VARCHAR2(100)

) ON COMMIT DELETE ROWS;

DECLARE

v\_codart product.product\_id%TYPE;

v\_precio price.list\_price%TYPE;

v\_descripcion product.description%TYPE;

BEGIN

-- Solicitar el código de artículo por teclado

v\_codart := &v\_codart; -- &v\_codart es una entrada del usuario

-- Verificar si el artículo existe

SELECT pr.list\_price, p.description

INTO v\_precio, v\_descripcion

FROM product p

JOIN price pr ON p.product\_id = pr.product\_id

WHERE p.product\_id = v\_codart AND ROWNUM = 1; -- Limitar a una fila

-- Si existe, realizar la inserción en la tabla temporal

INSERT INTO temp\_ej19 (codigo\_articulo, precio, descripcion)

VALUES (v\_codart, v\_precio, v\_descripcion);

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20001, 'El artículo (' || v\_codart || ') no lo ha pedido ningún cliente');

END;

/

1. Modificar el ejercicio 2 añadiendo excepciones, de tal forma que en el gestor de excepciones controlemos que select ha fallado, insertando en una tabla temporal cliente no existe, o bien articulo no existe, según haya fallado uno u otro.

**RESPUESTA:**

DROP TABLE temp\_ej20;

CREATE TABLE temp\_ej20 (mensaje VARCHAR2(100));

SET SERVEROUTPUT ON;

SET VERIFY OFF;

DECLARE

v\_customer\_id NUMBER(6);

v\_error\_msg VARCHAR2(100);

v\_count NUMBER(1);

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Ingrese el código de cliente: ');

v\_customer\_id := TO\_NUMBER(&v\_customer\_id);

BEGIN

SELECT COUNT(\*)

INTO v\_count

FROM customer

WHERE customer\_id = v\_customer\_id;

IF v\_count = 0 THEN

v\_error\_msg := 'El cliente no existe';

INSERT INTO temp\_ej20 (mensaje) VALUES (v\_error\_msg);

COMMIT;

END IF;

INSERT INTO temp2 (cod\_cli, nombre\_cli, tot\_ped, precio\_tot)

SELECT c.customer\_id, c.name, COUNT(so.order\_id) AS tot\_ped, SUM(so.total) AS precio\_tot

FROM customer c

LEFT JOIN sales\_order so ON c.customer\_id = so.customer\_id

WHERE c.customer\_id = v\_customer\_id

GROUP BY c.customer\_id, c.name;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

v\_error\_msg := 'El cliente no existe';

INSERT INTO temp\_ej20 (mensaje) VALUES (v\_error\_msg);

COMMIT;

WHEN OTHERS THEN

v\_error\_msg := 'Error en la consulta SELECT';

INSERT INTO temp\_ej20 (mensaje) VALUES (v\_error\_msg);

COMMIT;

END;

END;

/

1. Crear 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.

**RESPUESTA:**

CREATE OR REPLACE TRIGGER empleado\_borrado\_trigger

AFTER DELETE ON employee

FOR EACH ROW

DECLARE

v\_employee\_id employee.employee\_id%TYPE;

v\_employee\_name employee.first\_name%TYPE;

BEGIN

v\_employee\_id := :OLD.employee\_id;

v\_employee\_name := :OLD.first\_name;

DBMS\_OUTPUT.PUT\_LINE('Se ha eliminado el empleado con número: ' || v\_employee\_id || ' y nombre: ' || v\_employee\_name);

END;

/

1. Crear un procedimiento que reciba como parámetro un código de empleado, y modificar el salario de un empleado en función del número de empleados que tiene a su cargo:
   1. si no tiene ningún empleado a su cargo subirle 50 euros.
   2. si tiene 1 empleado a su cargo subirle 80 euros.
   3. si tiene 2 empleados a su cargo subirle 100 euros.
   4. si tiene más de tres empleados a su cargo subirle 110 euros.
   5. si es el PRESIDENTE su salario se incrementa en 30 euros.

**RESPUESTA:**

CREATE OR REPLACE PROCEDURE actualizar\_salario (

cod\_empl IN NUMBER

) IS

num\_empl NUMBER;

v\_slr\_incrm NUMBER;

BEGIN

-- Obtener el número de empleados a cargo

SELECT COUNT(\*)

INTO num\_empl

FROM employee

WHERE manager\_id = cod\_empl;

-- Verificar si es el presidente

IF cod\_empl = 1 THEN

v\_slr\_incrm := 30;

ELSE

-- Incrementar el salario según el número de empleados a cargo

CASE num\_empl

WHEN 0 THEN

v\_slr\_incrm := 50;

WHEN 1 THEN

v\_slr\_incrm := 80;

WHEN 2 THEN

v\_slr\_incrm := 100;

ELSE

v\_slr\_incrm := 110;

END CASE;

END IF;

-- Actualizar el salario del empleado

UPDATE employee

SET salary = salary + v\_slr\_incrm

WHERE employee\_id = cod\_empl;

COMMIT;

END;

/

1. Hacer dos bloques PL/SQL uno de los cuales escribe ‘Hola mundo’ al derecho y otro del revés.

**RESPUESTA:**

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Hola mundo');

END;

/

-- Hola mundo al revés

DECLARE

v\_linea VARCHAR2(50) := 'Hola mundo';

v\_lineaReves VARCHAR2(50);

BEGIN

FOR i IN REVERSE 1 .. LENGTH(v\_linea) LOOP

v\_lineaReves := v\_lineaReves || SUBSTR(v\_linea, i, 1);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE(v\_lineaReves);

END;

/

1. Hacer un bloque o proceso que muestre por pantalla el nombre y el jefe de cada empleado, con WHILE.

**RESPUESTA:**

DECLARE

v\_employee\_id employee.employee\_id%TYPE;

v\_employee\_name employee.first\_name%TYPE;

v\_manager\_name employee.first\_name%TYPE;

BEGIN

-- Declarar un cursor para obtener el nombre y el jefe de cada empleado

FOR emp\_rec IN (SELECT employee\_id, first\_name, manager\_id FROM employee) LOOP

v\_employee\_id := emp\_rec.employee\_id;

-- Obtener el nombre del empleado

SELECT first\_name INTO v\_employee\_name FROM employee WHERE employee\_id = v\_employee\_id;

-- Obtener el nombre del jefe del empleado

SELECT first\_name INTO v\_manager\_name FROM employee WHERE employee\_id = emp\_rec.manager\_id;

-- Imprimir el nombre y el jefe del empleado

DBMS\_OUTPUT.PUT\_LINE('Nombre: ' || v\_employee\_name || ', Jefe: ' || v\_manager\_name);

END LOOP;

END;

/

1. Visualizar los apellidos de los empleados pertenecientes al departamento pasado por parámetro, numerándolos secuencialmente.

**RESPUESTA:**

DECLARE

v\_dept\_id department.department\_id%TYPE := &v\_dept\_id;

v\_counter NUMBER := 1;

v\_last\_name employee.last\_name%TYPE;

CURSOR c\_emp IS

SELECT last\_name

FROM employee

WHERE department\_id = v\_dept\_id;

BEGIN

OPEN c\_emp;

LOOP

FETCH c\_emp INTO v\_last\_name;

EXIT WHEN c\_emp%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('[' || v\_counter || '] ' || v\_last\_name);

v\_counter := v\_counter + 1;

END LOOP;

CLOSE c\_emp;

END;

/

1. Escribir un procedimiento que reciba todos los datos de un nuevo empleado y procese la transacción de alta, gestionando los siguientes errores:
   1. no\_existe\_departamento
   2. no\_existe\_director
   3. numero\_empleado\_duplicado
   4. salario nulo con RAISE\_APPLICATION\_ERROR
   5. Otros posibles errores de Oracle visualizando código y mensaje de error.

**RESPUESTA:**

1. Escribe un procedimiento (recibe como parámetro el oficio) el procedimiento subirá el sueldo de todos los empleados que ganen menos del salario medio de su oficio. La subida será del 50% de la diferencia entre el salario de los empleados y la media de su oficio. Se deberá gestionar los posibles errores.

**RESPUESTA:**

1. Escribir un programa que incremente el salario de los empleados de un determinado departamento que se pasará como primer parámetro. El incremento será una cantidad en euros que se pasará como segundo parámetro en la llamada. El programa deberá informar del número de filas afectadas por la actualización. Los salarios se han de actualizar individualmente.

**RESPUESTA:**

1. Crear un programa que reciba un número de empleado y una cantidad que se incrementará al salario del empleado correspondiente. Utilizar una excepción definida por el usuario denominada salario\_nulo y la predefinida when no\_data\_found

**RESPUESTA:**