BASES DE DATOS – Parcial 2

Nombre: Alejandro Rodríguez López UO 281827 PL: 03

**INSTRUCCIONES**

La entrega de las soluciones será en formato PDF con nombre de fichero:

Ex2\_UOXXXXXX\_Nombre\_Apellido1.pdf

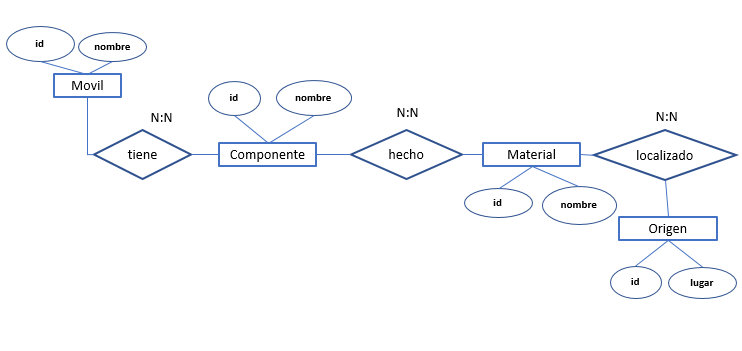
No incluir signos de puntuación en el nombre del fichero tales como acentos o diéresis.

Indicar en el apartado **RESPUESTA** la solución a cada ejercicio. En algunos casos se proporciona la salida esperada como resultado. **Importante:** Obtener la misma salida no es indicativo de obtener la máxima puntuación del ejercicio. Los nombres indicados en negrita de tablas, funciones, procedimientos, triggers, columnas, tuplas, etc. deben respetarse en la solución propuesta o bien son explicativos (en el caso de los parámetros formales de funciones y procedimientos se podrá elegir la denominación que se desee).

Nota: los ejercicios 1 y 2 deberían abordarse en secuencia pues son dependientes.

BD: **movil**

**MODELO E-R de la BD movil**

****

**Ejercicio 1 (1,5 puntos)**

Crear una tabla aislada, sin vínculos a otras tablas, en la base de datos **movil** llamada **operador**, popularizarla y codificar una función que opere sobre ella, todo conforme al siguiente detalle:

* Columnas: **id\_operador** (integer Primary Key); **nombre\_operador** (varchar(20) Not Null); **lineas\_operador** (integer Not Null).
* Popularizar con las tuplas: **(1,‘SKT’, 40000000)**; **(2,‘NTTDOCOMO’, 31120000)**; **(3,‘VERIZON’, 87450000)**; **(4,‘DT’, 37325000)**; **(5,‘TIM’, 28945000); (6,’CLARO’, 51637000).**
* Crear la función **operador\_lineas()** que proyecte la tabla **operador** por la columna **nombre\_operador** incluyendo solo aquellos operadores que tengan como máximo 38 millones de líneas y los que tengan como mínimo 80 millones de líneas.

RESPUESTA:

-- 1

-- 1.1 Crear tabla

CREATE TABLE operador (

id\_operador INTEGER PRIMARY KEY,

nombre\_operador VARCHAR(20) NOT NULL,

lineas\_operador INTEGER NOT NULL

);

-- 1.2 Popular tabla

INSERT INTO operador VALUES (1, 'SKT', 40000000);

INSERT INTO operador VALUES (2, 'NTTDOCOMO', 31120000);

INSERT INTO operador VALUES (3, 'VERIZON', 87450000);

INSERT INTO operador VALUES (4, 'DT', 37325000);

INSERT INTO operador VALUES (5, 'TIM', 28945000);

INSERT INTO operador VALUES (6, 'CLARO', 51637000);

-- Test

SELECT \* FROM operador;

-- 1.3 Funcion lineas <= 38M o lineas >= 80M

CREATE OR REPLACE FUNCTION operador\_lineas () RETURNS TABLE (nombre\_operador VARCHAR(20)) AS $$

BEGIN

RETURN QUERY

SELECT operador.nombre\_operador

FROM operador

WHERE operador.lineas\_operador <= 38000000

OR operador.lineas\_operador >= 80000000;

END;

$$ LANGUAGE PLPGSQL;

-- Test

SELECT operador\_lineas();

**Ejercicio 2 (2 puntos)**

La tabla **operador** está aislada en la base de datos **movil** porque la utilizamos como referencia documental acumulada, por políticas de nuestra compañía incluiremos una restricción a las operaciones de borrado mediane la operación combinada de un trigger y su función asociada: codificar el trigger **delete\_operador\_trigger** y la función asociada **delete\_operador()** de forma que operen antes de un borrado en la tabla **operador** impidiendo llevar a término la operación solo en el caso, el resto de supuestos sí se llevarían a término, de que se intente borrar un registro con al menos 35 millones de líneas en la columna **líneas\_operador**, si fuera el caso además de impedir la operación se imprimirá por consola el siguiente mensaje **‘operación no autorizada consulte al administrador de la base de datos’**.

RESPUESTA:

-- 2

-- 2.1 Funcion abortar si delete y lineas >= 35M

CREATE OR REPLACE FUNCTION delete\_operador () RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'DELETE' AND old.lineas\_operador >= 35000000

THEN

RAISE NOTICE 'Operacion no autorizada, consulte al administrador de la base de datos.';

RETURN NULL;

END IF;

RETURN new;

END;

$$ LANGUAGE PLPGSQL;

-- 2.2 Trigger antes de delete

CREATE TRIGGER delete\_operador\_trigger

BEFORE DELETE ON operador

FOR EACH ROW EXECUTE PROCEDURE delete\_operador();

-- test

INSERT INTO operador VALUES (7, 'Alex', 50000000);

INSERT INTO operador VALUES (8, 'Alex', 27);

DELETE FROM operador WHERE operador.id\_operador=7;

DELETE FROM operador WHERE operador.id\_operador=8;

DROP TRIGGER delete\_operador\_trigger

ON operador;

DELETE FROM operador WHERE operador.id\_operador=7;

CREATE TRIGGER delete\_operador\_trigger

BEFORE DELETE ON operador

FOR EACH ROW EXECUTE PROCEDURE delete\_operador();

**Ejercicio 3 (2 puntos)**

Codificar el trigger **deup\_origen\_trigger** y la función asociada **deup\_origenl()** que operen como sigue: tras una operación de inserción o borrado en la tabla **origen** se imprimirá por consola un mensaje específico que indique el tipo de operación que se ha realizado junto con el **id** del registro (valor de la columna **id** de la tabla **origen**) implicado en la operación.

RESPUESTA:

-- 3

-- 3.1 Funcion mostrar operacion e id

CREATE OR REPLACE FUNCTION deup\_original () RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'DELETE'

THEN

-- Delete ==> Return old

RAISE NOTICE 'Se ha realizado una operacion % en la tabla ''origen'', en el id %.', TG\_OP, old.id;

RETURN old;

ELSE

-- Non-Delete ==> Return new

RAISE NOTICE 'Se ha realizado una operacion % en la tabla ''origen'', en el id %.', TG\_OP, new.id;

RETURN new;

END IF;

END;

$$ LANGUAGE PLPGSQL;

-- 3.2 Trigger tras delete o insert

CREATE TRIGGER deup\_original\_trigger

AFTER INSERT OR DELETE ON origen

FOR EACH ROW EXECUTE PROCEDURE deup\_original ();

-- test

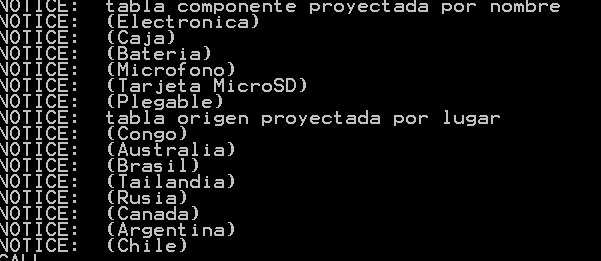
INSERT INTO origen VALUES (309, 'Espanna');

DELETE FROM origen WHERE id=309;

**Ejercicio 4 (2,25 puntos)**

Codificar un procedimiento **listar\_co\_proyectados()** que opere como sigue: imprimirá por consola y en secuencia el contenido de las tablas **componente** y **origen**, proyectadas respectivamente por las columnas **nombre** y **lugar**, además ambos listados estarán precedidos, también respectivamente, por los textos **‘tabla componente proyectada por nombre’** y **‘tabla origen proyectada por lugar’**.

La siguiente captura se corresponde a la salida de la invocación **listar\_co\_proyectados()**



RESPUESTA:

-- 4

-- 4.1 Procedimiento listar nombre de componentes y lugares de origenes

CREATE OR REPLACE PROCEDURE listar\_co\_proyectados () AS $$

DECLARE R VARCHAR(20);

BEGIN

-- nombres de componentes

RAISE NOTICE 'Tabla ''componente'' proyectada por ''nombre''';

FOR R IN

SELECT componente.nombre

FROM componente

LOOP

RAISE NOTICE '%', R;

END LOOP;

-- lugares de origenes

RAISE NOTICE 'Tabla ''origen'' proyectada por ''lugar''';

FOR R IN

SELECT origen.lugar

FROM origen

LOOP

RAISE NOTICE '%', R;

END LOOP;

END;

$$ LANGUAGE PLPGSQL;

-- test

CALL listar\_co\_proyectados();

**Ejercicio 5 (2,25 puntos)**

Codificar una función **apple()** que incluya una consulta SQL cuya salida se corresponderá con la que aparece en la captura de pantalla resultado de su invocación (el orden intratupla debe respetarse, el orden intertupla es indiferente).

(Nota: durante el proceso de inferencia de la consulta solicitada se recomienda consultar la arquitectura de la base de datos **movil** y el detalle de estructura de las tablas que la integran)



RESPUESTA:

-- 5

-- 5.1 Funcion nombreMovil idMovil componenteId componenteNombre de 'apple'

CREATE OR REPLACE FUNCTION apple () RETURNS TABLE (nombre\_movil VARCHAR(20), id\_movil INTEGER, id\_componente INTEGER, nombre\_componente VARCHAR(20)) AS $$

BEGIN

RETURN QUERY

SELECT movil.nombre, movil.id, componente.id, componente.nombre

FROM movil

JOIN movilComponente ON movil.id = movilComponente.id\_movil

JOIN componente ON movilComponente.id\_componente = componente.id

WHERE LOWER(movil.nombre) = 'apple';

END;

$$ LANGUAGE PLPGSQL;

-- test

SELECT apple ();