Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Administración y Diseño de Bases de Datos:

Modelo Lógico Relacional (Disparadores): Viveros

Anabel Díaz Labrador
(alu0101206011@ull.edu.es)

Andrea Calero Caro
(alu0101202952@ull.edu.es)

Sheyla Ruiz-Gómez Ferreira
(alu0101124445@ull.edu.es)

Alejandro Martín de León
(alu0101015941@ull.edu.es

1. Detalles sobre los disparadores creados

En primer lugar, para el caso del disparador número uno, se ha creado la siguiente estructura:

```
CREATE OR REPLACE FUNCTION crear_email() RETURNS trigger AS $trigger_crear_email_before_insert$
    BEGIN
        IF NEW.Nombre IS NULL THEN
            RAISE EXCEPTION 'Nombre no puede ser nulo';
        END IF;
        IF NEW.Apellidos IS NULL THEN
            RAISE EXCEPTION 'Apellidos no pueden ser nulo';
        END IF;

        IF (NEW.Email IS NULL OR NEW.Email = '') THEN
            NEW.Email := CONCAT(NEW.Nombre, NEW.Apellidos, '@', 'gmail.com');

        END IF;
        RETURN NEW;
    END;
$trigger_crear_email_before_insert$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trigger_crear_email_before_insert BEFORE INSERT ON CLIENTES_PLUS
FOR EACH ROW EXECUTE PROCEDURE crear_email();
```

Además, para cumplir con las especificaciones que se piden para la creación del email, se ha incluido en el método el comando “CHECK”, para comprobar de que el email introducido sigue las normas de nomenclatura (un conjunto de caracteres seguido del carácter “@” y finalizado por el dominio). Se puede ver a continuación la parte del código que realiza dicha comprobación:

```
DROP TABLE IF EXISTS CLIENTES_PLUS CASCADE;

CREATE TABLE IF NOT EXISTS CLIENTES_PLUS (
            DNI INT NOT NULL,
            Nombre VARCHAR(30),
            Apellidos VARCHAR(45),
            Total_Mensual FLOAT NULL,
            Bonificacion FLOAT NULL,
            EMPLEADOS_DNI INT NOT NULL,
            Email VARCHAR(45),
            CONSTRAINT email CHECK (Email ~* '^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'),
            PRIMARY KEY (DNI),
            CONSTRAINT fk_CLIENTES_PLUS_EMPLEADOS1
              FOREIGN KEY (EMPLEADOS_DNI)
              REFERENCES EMPLEADOS (DNI)
              ON DELETE NO ACTION
              ON UPDATE NO ACTION);
```

Para el segundo apartado, se ha interpretado que en cada municipio solo pudiera haber un vivero. Para esto, se ha creado la tabla MUNICIPIO, la cual contiene como clave ajena las coordenadas geográficas de VIVEROS, de tal forma que en la función creada a posteriori, compruebe si:

- El municipio ha sido creado con anterioridad.
- Si los nombres coinciden.

Para esto se ha creado un trigger que comprueba antes de insertar.

```
DROP TABLE IF EXISTS MUNICIPIO CASCADE;

CREATE TABLE IF NOT EXISTS MUNICIPIO (

            Código_postal INT NOT NULL,
            Nombre VARCHAR(45) NOT NULL,
            Superficie VARCHAR(45) NULL,
            VIVEROS_Coordenadas_Geograficas FLOAT NOT NULL,
            UNIQUE (Nombre),
            PRIMARY KEY (Código_postal, VIVEROS_Coordenadas_Geograficas),
          CONSTRAINT fk_MUNICIPIO_VIVEROS1
            FOREIGN KEY (VIVEROS_Coordenadas_Geograficas)
            REFERENCES VIVEROS (Coordenadas_Geograficas)
            ON DELETE NO ACTION
            ON UPDATE NO ACTION);
```

```
CREATE OR REPLACE FUNCTION municipio_existe() RETURNS trigger AS $trigger_municipio_existe$
  BEGIN
  IF (NEW.Nombre = OLD.Nombre) THEN
    RAISE EXCEPTION 'El Municipio ya fue registrado con anterioridad';
  END IF;
  RETURN NEW;
  END;
$trigger_municipio_existe$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trigger_municipio_existe BEFORE INSERT ON MUNICIPIO
FOR EACH ROW EXECUTE PROCEDURE municipio_existe();
```

Como iniciativa propia se ha creado un disparador que verifique el número de días de vacaciones que tiene un empleado. Primero, se añadió una nueva columna en

la tabla EMPLEADOS, que es el número de días de vacaciones. Luego, se comprueba si el empleado tiene o no el número de días de vacaciones asignado. En caso de no disponer asociado ningún día, se le adjudica el mínimo (22 días). Mientras que si los tiene, verifica que no sea inferior a 0 ni superior a 60 días. Siguiendo la siguiente estructura:

```
CREATE OR REPLACE FUNCTION verificar_vacaciones() RETURNS trigger AS $trigger_verificar_vacaciones$
    BEGIN
        IF NEW.Vacaciones_anuales IS NULL THEN
            NEW.Vacaciones_anuales = 22;
        ELSIF NEW.Vacaciones_anuales > 60 THEN
          RAISE EXCEPTION 'Se han excedido los días de vacaciones';
        ELSIF NEW.Vacaciones_anuales < 0 THEN
          RAISE EXCEPTION 'Los días de vacaciones no pueden ser un valor negativo.';
        END IF;
        RETURN NEW;
    END;
$trigger_verificar_vacaciones$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trigger_verificar_vacaciones BEFORE INSERT ON EMPLEADOS
FOR EACH ROW EXECUTE PROCEDURE verificar_vacaciones();
```

Por último, para el caso del tercer apartado, se ha creado un disparador que permite mantener de forma actualizada el stock de los productos del vivero. Para este caso, la tabla PRODUCTO tiene asociado el atributo **stock**, el cuál será actualizado a la hora de realizar una inserción en la tabla PEDIDO. A la cantidad original del producto, se le resta el número de ese mismo producto comprado por un cliente. A continuación, se detalla la parte del código que permitirá llevar a cabo esta tarea:

```
CREATE OR REPLACE FUNCTION actualizar_stock_clientes() RETURNS trigger AS $trigger_actualizar_stock_clientes$
    BEGIN
        UPDATE PRODUCTOS SET
          Stock = Stock - NEW.Cantidad
          WHERE NEW.PRODUCTOS_Codigo_Producto = Codigo_Producto;

        RETURN NEW;
    END;
$trigger_actualizar_stock_clientes$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION actualizar_stock_productos() RETURNS trigger AS $trigger_actualizar_stock_productos$
    BEGIN
        IF EXISTS (SELECT * FROM PRODUCTOS WHERE NEW.Codigo_Producto = Codigo_Producto) THEN
          UPDATE PRODUCTOS SET
            Stock = Stock + NEW.Stock
            WHERE NEW.Codigo_Producto = Codigo_Producto;
            RAISE NOTICE 'Actualizando stock';
            RETURN NULL;
        END IF;
        RETURN NEW;
    END;
$trigger_actualizar_stock_productos$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trigger_actualizar_stock_clientes AFTER INSERT ON CLIENTES_PLUS_PRODUCTOS
FOR EACH ROW EXECUTE PROCEDURE actualizar_stock_clientes();

CREATE TRIGGER trigger_actualizar_stock_productos BEFORE INSERT ON PRODUCTOS
FOR EACH ROW EXECUTE PROCEDURE actualizar_stock_productos();
```

2. SELECT de cada tabla de la base de datos

1.1. Tabla PRODUCTOS:

```
INSERT 0 1
 codigo_producto | nombre  | precio | stock
-----------------+---------+--------+-------
               0 | Lechuga |      1 |    50
               1 | Tomates |      1 |    70
(2 rows)

INSERT 0 1
 codigo_producto | nombre  | precio | stock
-----------------+---------+--------+-------
               1 | Tomates |      1 |    70
               0 | Lechuga |      1 |    40
(2 rows)

psql:script:236: NOTICE:  Actualizando stock
INSERT 0 0
 codigo_producto | nombre  | precio | stock
-----------------+---------+--------+-------
               1 | Tomates |      1 |    70
               0 | Lechuga |      1 |  4040
(2 rows)
```

Los primeros inserts son ‘Lechuga’ y ‘Tomates’ con la tabla vacía.
El segundo insert es un pedido de 10 lechugas.
En el tercero se intentan introducir 4000 lechugas.

1.2. Tabla CLIENTES_PLUS.

```
    dni    | nombre | apellidos | total_mensual | bonificacion | empleados_dni |         email
-----------+--------+-----------+---------------+--------------+---------------+-----------------------
 790638369 | Anabel | Diaz      |         45753 |          0.3 |     123456789 | anabel@diaz.es
 790643556 | Martin | Martin    |         30000 |          0.4 |     123456789 | MartinMartin@gmail.com
 788512343 | Jaime  | Pablo     |          4000 |          0.1 |     123456789 | JaimePablo@gmail.com
(3 rows)
```

1.3. Tabla VIVEROS.

```
 coordenadas_geograficas |  nombre  | superficie
-------------------------+----------+------------
                    32.3 | Tenerife | 400000
(1 row)
```

1.4. Tabla ZONAS.

```
 coordenadas_geograficas | nombre
-------------------------+---------
                    32.3 | Regadío
(1 row)
```

1.5. Tabla CLIENTES_PLUS_PRODUCTOS.

```
 codigo_pedido | clientes_plus_dni | productos_codigo_producto | cantidad |   fecha
---------------+-------------------+---------------------------+----------+------------
             1 |         790638369 |                         0 |       10 | 2021-11-11
(1 row)
```

1.6. Tabla ZONAS_PRODUCTOS.

```
 nombre  | productos_codigo_producto | zonas_coordenadas_geograficas
---------+---------------------------+-------------------------------
 Regadío |                         0 |                          32.3
(1 row)
```

1.7 Tabla MUNICIPIO

```
 código_postal |  nombre   | superficie | viveros_coordenadas_geograficas
---------------+-----------+------------+---------------------------------
         34567 | Tacoronte | 3000       |                            32.3
 (1 row)
```