


	CONSTRUCCIÓN DEL SISTEMA DE INFORMACIÓN (CSI)	BASES DE DATOS
 Escuela Superior de Ingeniería y Tecnología Universidad de La Laguna	PROYECTO: Sistema de bases de datos para empresa logística	Generación de código
	Autor: Jesús Carmelo González Domínguez (alu0101267760@ull.edu.es) Carlota Marrero Morales (alu0101060385@ull.edu.es) Aday Padilla Amaya (alu0100843453@ull.edu.es) Claudio Néstor Yanes Mesa (alu0101229942@ull.edu.es)	
Versión: 1.0	Tiempo invertido: 9h	Fecha: 06/02/22

CÓDIGO DE COMPONENTES
<pre> CREATE TABLE IF NOT EXISTS tipos_edificio(id SMALLINT PRIMARY KEY, tipo VARCHAR NOT NULL UNIQUE); CREATE TABLE IF NOT EXISTS edificio(edificio_id SERIAL PRIMARY KEY, nombre VARCHAR NOT NULL UNIQUE, direccion VARCHAR NOT NULL, tipo SMALLINT NOT NULL, FOREIGN KEY (tipo) REFERENCES tipos_edificio(id)); CREATE TABLE IF NOT EXISTS departamento(departamento_id SERIAL PRIMARY KEY, nombre VARCHAR NOT NULL UNIQUE); CREATE TABLE IF NOT EXISTS roles_empleado(id SMALLINT PRIMARY KEY, rol VARCHAR NOT NULL UNIQUE); CREATE TABLE IF NOT EXISTS empleado(dni CHAR(9) PRIMARY KEY, nombre_completo VARCHAR NOT NULL, iban CHAR(34) NOT NULL, rol SMALLINT NOT NULL, departamento_id INT NOT NULL, edificio_id INT NOT NULL, FOREIGN KEY (rol) REFERENCES roles_empleado(id), FOREIGN KEY (departamento_id) REFERENCES departamento(departamento_id), FOREIGN KEY (edificio_id) REFERENCES edificio(edificio_id)); CREATE TABLE IF NOT EXISTS turno(dni CHAR(9) NOT NULL, comienzo TIMESTAMP NOT NULL, final TIMESTAMP NOT NULL, horas_trabajadas numeric, CHECK (comienzo < final), </pre>

```

        PRIMARY KEY (dni, comienzo),
        FOREIGN KEY (dni) REFERENCES empleado(dni) ON DELETE CASCADE ON UPDATE
CASCADE
);

CREATE TABLE IF NOT EXISTS cliente(
    cliente_id SERIAL PRIMARY KEY,
    nombre VARCHAR NOT NULL,
    telefono VARCHAR NOT NULL,
    email VARCHAR NOT NULL UNIQUE,
    gestor CHAR(9) NOT NULL,
    CHECK (email ~* '^[A-Z0-9._%~+@[A-Z0-9.-]+\.[A-Z]{2,4}$'), -- Regex from
https://emailregex.com/
    FOREIGN KEY (gestor) REFERENCES empleado(dni)
);

CREATE TABLE IF NOT EXISTS estado_vehiculo(
    id SMALLINT PRIMARY KEY,
    estado VARCHAR NOT NULL
);

CREATE TABLE IF NOT EXISTS vehiculo(
    matricula CHAR(8) PRIMARY KEY,
    estado SMALLINT NOT NULL,
    FOREIGN KEY (estado) REFERENCES estado_vehiculo(id)
);

CREATE TABLE IF NOT EXISTS chofer_conduce_vehiculo(
    dni CHAR(9) NOT NULL,
    matricula CHAR(8) NOT NULL,
    comienzo TIMESTAMP NOT NULL,
    final TIMESTAMP,
    CHECK (comienzo < final),
    PRIMARY KEY (dni, matricula, comienzo),
    FOREIGN KEY (dni) REFERENCES empleado(dni) ON DELETE CASCADE ON UPDATE
CASCADE,
    FOREIGN KEY (matricula) REFERENCES vehiculo(matricula) ON DELETE CASCADE ON
UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS ruta(
    ruta_id SERIAL PRIMARY KEY,
    fecha DATE NOT NULL,
    ruta BYTEA,
    matricula CHAR(8),
    FOREIGN KEY (matricula) REFERENCES vehiculo(matricula)
);

CREATE TABLE IF NOT EXISTS factura(
    factura_id SERIAL PRIMARY KEY,
    fecha_de_emision DATE NOT NULL,
    fecha_de_pago DATE,
    total NUMERIC(12, 2) NOT NULL DEFAULT 0.00,
    CHECK (fecha_de_emision <= fecha_de_pago)
);

CREATE TABLE IF NOT EXISTS servicio(
    servicio_id SERIAL PRIMARY KEY,
    punto de recogida VARCHAR NOT NULL,

```

```

    punto_de_entrega VARCHAR NOT NULL,
    fecha_de_recogida DATE NOT NULL,
    fecha_de_entrega DATE NOT NULL,
    requisitos_especiales VARCHAR,
    cliente_id INT NOT NULL,
    factura_id INT NOT NULL,
    coste NUMERIC(12, 2) NOT NULL,
    CHECK (fecha_de_recogida <= fecha_de_entrega),
    FOREIGN KEY (cliente_id) REFERENCES cliente(cliente_id),
    FOREIGN KEY (factura_id) REFERENCES factura(factura_id)
);

CREATE TABLE IF NOT EXISTS estados_pale(
    id SMALLINT PRIMARY KEY,
    estado VARCHAR NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS pale(
    pale_id SERIAL NOT NULL,
    servicio_id INT NOT NULL,
    estado SMALLINT NOT NULL,
    PRIMARY KEY (pale_id, servicio_id),
    FOREIGN KEY (estado) REFERENCES estados_pale(id),
    FOREIGN KEY (servicio_id) REFERENCES servicio(servicio_id) ON DELETE
    CASCADE ON UPDATE CASCADE
);

CREATE TABLE ruta_distribuye_pale(
    ruta_id INT NOT NULL,
    pale_id INT NOT NULL,
    servicio_id INT NOT NULL,
    PRIMARY KEY (ruta_id, pale_id, servicio_id),
    FOREIGN KEY (ruta_id) REFERENCES ruta(ruta_id) ON DELETE CASCADE ON UPDATE
    CASCADE,
    FOREIGN KEY (pale_id, servicio_id) REFERENCES pale(pale_id, servicio_id) ON
    DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE almacen_almacena_pale(
    edificio_id INT NOT NULL,
    pale_id INT NOT NULL,
    servicio_id INT NOT NULL,
    comienzo TIMESTAMP NOT NULL,
    final TIMESTAMP,
    CHECK (comienzo < final),
    PRIMARY KEY (edificio_id, pale_id, servicio_id, comienzo),
    FOREIGN KEY (edificio_id) REFERENCES edificio(edificio_id) ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY (pale_id, servicio_id) REFERENCES pale(pale_id, servicio_id) ON
    DELETE CASCADE ON UPDATE CASCADE
);

-- Inicializa "enums"

INSERT INTO tipos_edificio(id, tipo) VALUES (0, 'oficina');
```

```

INSERT INTO tipos_edificio(id, tipo) VALUES (1, 'almacén');

INSERT INTO roles_empleado(id, rol) VALUES (0, 'administrativo');
INSERT INTO roles_empleado(id, rol) VALUES (1, 'operador');
INSERT INTO roles_empleado(id, rol) VALUES (2, 'chófer');

INSERT INTO estado_vehiculo(id, estado) VALUES (0, 'en servicio');
INSERT INTO estado_vehiculo(id, estado) VALUES (1, 'inoperativo');

INSERT INTO estados_pale(id, estado) VALUES (0, 'registrado');
INSERT INTO estados_pale(id, estado) VALUES (1, 'en transito');
INSERT INTO estados_pale(id, estado) VALUES (2, 'dañado');
INSERT INTO estados_pale(id, estado) VALUES (3, 'entregado');

```

CÓDIGO DE PROCEDIMIENTOS DE OPERACIÓN Y SEGURIDAD

```

-- Funciones y triggers

-- La palabra clave de POSTGRESQL ASSERT resulta muy util para crear triggers que comprueben la integridad de los
datos.
-- No obstante, ASSERT puede ser desactivado con plpgsql.check_asserts, por lo que se ha optado por crear una
función
-- no desactivable que imita el comportamiento de ASSERT.
CREATE OR REPLACE FUNCTION __assert(condition boolean, msg varchar default '__assertion failed') RETURNS VOID
LANGUAGE plpgsql AS $$
BEGIN
    IF NOT condition THEN
        RAISE EXCEPTION '%', msg;
    END IF;
END;
$$;

CREATE OR REPLACE FUNCTION validar_empleado() RETURNS TRIGGER LANGUAGE plpgsql AS $$
BEGIN
    IF (NEW.rol = 0) THEN -- administrativo
        PERFORM __assert((SELECT tipo FROM edificio WHERE edificio_id = NEW.edificio_id) = 0, 'los
administrativo solo pueden asignarse a oficinas');
    ELSIF (NEW.rol = 1 OR NEW.rol = 2) THEN -- Operario o chofer
        PERFORM __assert((SELECT tipo FROM edificio WHERE edificio_id = NEW.edificio_id) = 1, 'los operarios
y choferes solo pueden ser asignados a almacenes');
    ELSE
        RAISE EXCEPTION 'empleado con rol no esperado, por favor, contacte con el administrador';
    END IF;
    RETURN NEW;
END;
$$;

DROP TRIGGER IF EXISTS validar_empleado_trigger ON empleado;
CREATE TRIGGER validar_empleado_trigger BEFORE INSERT OR UPDATE ON empleado FOR EACH ROW EXECUTE PROCEDURE
validar_empleado();

CREATE OR REPLACE FUNCTION validar_turno() RETURNS TRIGGER LANGUAGE plpgsql AS $$
DECLARE
    comienzo_previo TIMESTAMP;
BEGIN
    IF tg_op = 'UPDATE' THEN
        comienzo_previo := OLD.comienzo;
    ELSIF tg_op = 'INSERT' then
        comienzo_previo := NEW.comienzo;
    END IF;
    PERFORM __assert(NOT EXISTS(SELECT 1 FROM turno WHERE dni = NEW.dni
                                AND comienzo != comienzo_previo
                                AND (comienzo, final)
                                OVERLAPS (NEW.comienzo, NEW.final)),
        'dos turnos de un mismo empleado no pueden solaparse');
    NEW.horas_trabajadas = (EXTRACT(EPOCH FROM NEW.final) - EXTRACT(EPOCH FROM NEW.comienzo))/3600;
    RETURN NEW;
END;
$$;

DROP TRIGGER IF EXISTS validar_turno_trigger ON turno;
CREATE TRIGGER validar_turno_trigger BEFORE INSERT OR UPDATE ON turno FOR EACH ROW EXECUTE PROCEDURE
validar_turno();

CREATE OR REPLACE FUNCTION validar_cliente() RETURNS TRIGGER LANGUAGE plpgsql AS $$
BEGIN

```

```

        PERFORM __assert((SELECT rol FROM empleado WHERE dni = NEW.gestor) = 0, 'un cliente solo puede ser
gestionado por un administrativo');
        RETURN NEW;
    END;
$$;
DROP TRIGGER IF EXISTS validar_cliente_trigger ON cliente;
CREATE TRIGGER validar_cliente_trigger BEFORE INSERT OR UPDATE ON cliente
    FOR EACH ROW EXECUTE PROCEDURE validar_cliente();

CREATE OR REPLACE FUNCTION validar_conduce() RETURNS TRIGGER LANGUAGE plpgsql AS $$
DECLARE
    n_final TIMESTAMP;
BEGIN
    n_final := CASE WHEN NEW.final IS NULL THEN now() ELSE NEW.final END;
    PERFORM __assert(NOT EXISTS(SELECT 1 FROM chofer_conduce_vehiculo WHERE matricula = NEW.matricula AND
                                comienzo != NEW.comienzo AND
                                (comienzo, CASE WHEN final IS NULL THEN now() ELSE final
                                OVERLAPS (NEW.comienzo, n_final))),
        'un vehiculo no puede tener dos chofers a la vez');
    RETURN NEW;
END;
$$;
DROP TRIGGER IF EXISTS validar_conduce_trigger ON chofer_conduce_vehiculo;
CREATE TRIGGER validar_conduce_trigger AFTER INSERT OR UPDATE ON chofer_conduce_vehiculo
    FOR EACH ROW EXECUTE PROCEDURE validar_conduce();

CREATE OR REPLACE FUNCTION validar_servicio() RETURNS TRIGGER LANGUAGE plpgsql AS $$
DECLARE
    id_cliente int;
BEGIN
    id_cliente := (SELECT DISTINCT cliente_id FROM servicio WHERE factura_id = NEW.factura_id);
    IF id_cliente IS NULL THEN
        UPDATE factura SET total = NEW.coste WHERE factura_id = NEW.factura_id;
        RETURN NEW;
    END IF;
    PERFORM __assert(id_cliente = NEW.cliente_id,
        'los servicios agrupados en una misma factura deben pertenecer al mismo cliente');
    IF tg_op = 'UPDATE' AND OLD.factura_id != NEW.factura_id THEN
        UPDATE factura SET total = COALESCE((SELECT SUM(coste) FROM servicio WHERE factura_id =
OLD.factura_id), 0.00)
        WHERE factura_id = OLD.factura_id;
    END IF;
    UPDATE factura SET total = COALESCE((SELECT SUM(coste) FROM servicio WHERE factura_id =
NEW.factura_id),
        0.00)
        WHERE factura_id = NEW.factura_id;
    RETURN NEW;
END;
$$;
DROP TRIGGER IF EXISTS validar_servicio_trigger ON servicio;
CREATE TRIGGER validar_servicio_trigger AFTER INSERT OR UPDATE ON servicio
    FOR EACH ROW EXECUTE PROCEDURE validar_servicio();

CREATE OR REPLACE FUNCTION recalcular_tras_eliminar_servicio() RETURNS TRIGGER LANGUAGE plpgsql AS $$
BEGIN
    UPDATE factura SET total = COALESCE((SELECT SUM(coste) FROM servicio WHERE factura_id = OLD.factura_id),
0.00)
    WHERE factura_id = OLD.factura_id;
    RETURN OLD;
END;
$$;
DROP TRIGGER IF EXISTS recalcular_tras_eliminar_servicio_trigger ON servicio;
CREATE TRIGGER recalcular_tras_eliminar_servicio_trigger AFTER DELETE ON servicio
    FOR EACH ROW EXECUTE PROCEDURE recalcular_tras_eliminar_servicio();

CREATE OR REPLACE FUNCTION validar_pale() RETURNS TRIGGER LANGUAGE plpgsql AS $$
BEGIN
    IF (NEW.estado = 3) THEN
        PERFORM __assert(NOT EXISTS(SELECT 1 FROM almacen_almacena_pale WHERE pale_id = NEW.pale_id
                                AND servicio_id = NEW.servicio_id AND (final IS NULL OR final > now()))),
        'un pale que esta almacenado no puede marcarse como entregado');
    END IF;
    RETURN NEW;
END;
$$;
DROP TRIGGER IF EXISTS validar_pale_trigger ON pale;
CREATE TRIGGER validar_pale_trigger AFTER INSERT OR UPDATE ON pale FOR EACH ROW EXECUTE PROCEDURE validar_pale();

CREATE OR REPLACE FUNCTION validar_almacen_almacena_pale() RETURNS TRIGGER LANGUAGE plpgsql AS $$

```

```

DECLARE
    n_final TIMESTAMP;
BEGIN
    PERFORM __assert((SELECT tipo FROM edificio WHERE edificio_id = NEW.edificio_id) = 1,
        'un pale solo se puede almacenar en un almacén');
    PERFORM __assert(NOT EXISTS(SELECT 1 FROM pale WHERE pale_id = NEW.pale_id AND
        servicio_id = NEW.servicio_id AND
        estado IN (0, 3)),
        'un pale en estado registrado o entregado no puede ser almacenado');

    n_final := CASE WHEN NEW.final IS NULL THEN now() ELSE NEW.final END;
    PERFORM __assert(NOT EXISTS(SELECT 1 FROM almacen_almacena_pale WHERE pale_id = NEW.pale_id AND
        servicio_id = NEW.servicio_id AND
        (comienzo != NEW.comienzo OR edificio_id != NEW.edificio_id) AND
        (comienzo, CASE WHEN final IS NULL THEN now() ELSE final END)
        OVERLAPS (NEW.comienzo, n_final)),
        'un pale no puede estar almacenado dos veces a la vez');

    RETURN NEW;
END;
$$;

DROP TRIGGER IF EXISTS validar_almacen_almacena_pale_trigger ON almacen_almacena_pale;
CREATE TRIGGER validar_almacen_almacena_pale_trigger AFTER INSERT OR UPDATE ON almacen_almacena_pale
    FOR EACH ROW EXECUTE PROCEDURE validar_almacen_almacena_pale();

```