

Informe Práctica

Administración
De bases de datos

Sergio Perera Márquez
alu0101394503@ull.edu.es

Brayan José Pérez Ramos
alu0101431185@ull.edu.es



Índice

Introducción y objetivos.....	3
Requisitos.....	3
Diagrama entidad-relación.....	5
Restricciones semánticas.....	5
Diagrama relacional.....	6
Base de datos SQL.....	7
Tablas base y tipos.....	7
Tipos.....	7
Tipo_producto.....	7
Tipo_zona.....	7
Local.....	7
Zona.....	8
Producto.....	8
Zona_producto.....	9
Empleado.....	9
Socio.....	10
Pedido.....	10
Trabaja.....	12
Pedido_producto.....	12
Libre.....	13
Herencia.....	13
Sobremesa.....	13
Portatiles.....	14
Dispositivos_moviles.....	14
Consola.....	15
Inclusividad.....	15
Empleado-trabaja.....	15
Pedido-trabaja.....	16
Ejemplos de consultas.....	17
SELECT.....	17
INSERT.....	17
Local.....	17
Zona.....	18
Producto.....	19



Zona_producto.....	20
Empleado.....	21
Socio.....	22
Trabaja.....	22
Pedido.....	23
Pedido_producto.....	24
Libre.....	25
Sobremesa.....	25
Portatiles.....	26
Dispositivos_moviles.....	26
Consola.....	26
UPDATE.....	27
Local.....	27
Zona.....	27
Producto.....	28
Zona_producto.....	28
Empleado.....	29
Socio.....	29
Trabaja.....	30
Pedido.....	30
Pedido_producto.....	30
Libre.....	31
Sobremesa.....	31
Portatiles.....	31
Dispositivos_moviles.....	32
Consola.....	32
DELETE.....	32
Local.....	32
Zona.....	33
Producto.....	33
Zona_producto.....	33
Empleado.....	33
Socio.....	33
Trabaja.....	33
Pedido.....	33
Pedido_producto.....	33
Libre.....	34
Sobremesa.....	34
Portatiles.....	34
Dispositivos_moviles.....	34
Consola.....	34



Implementación de API.....	35
Operaciones CRUD.....	35
Estructuración de las rutas.....	35
Pruebas de API.....	36
GET.....	36
GET por id.....	36
GET general.....	37
POST.....	39
PUT.....	40
DELETE.....	42
Presupuesto.....	43
Conclusiones.....	43



Introducción y objetivos

En este proyecto se centra en el diseño, implementación y gestión de una base de datos relacional para una entidad comercial ficticia, abarcando diferentes áreas clave del negocio, como el inventario de productos, la administración de empleados, la interacción con socios y clientes, y la distribución de productos en distintas zonas.

El objetivo principal de este proyecto es crear un modelo de base de datos que no solo permita almacenar información de manera estructurada y eficiente, sino también facilite su consulta y actualización para apoyar las operaciones diarias de la entidad comercial. Para ello, se ha diseñado una base de datos con catorce tablas interrelacionadas que abarcan aspectos como la gestión de productos, pedidos, empleados, zonas de distribución y relaciones entre estos elementos.

Además, el proyecto busca optimizar el acceso y la manipulación de los datos mediante la implementación de índices, restricciones y triggers que validan automáticamente la consistencia de la información.

Requisitos

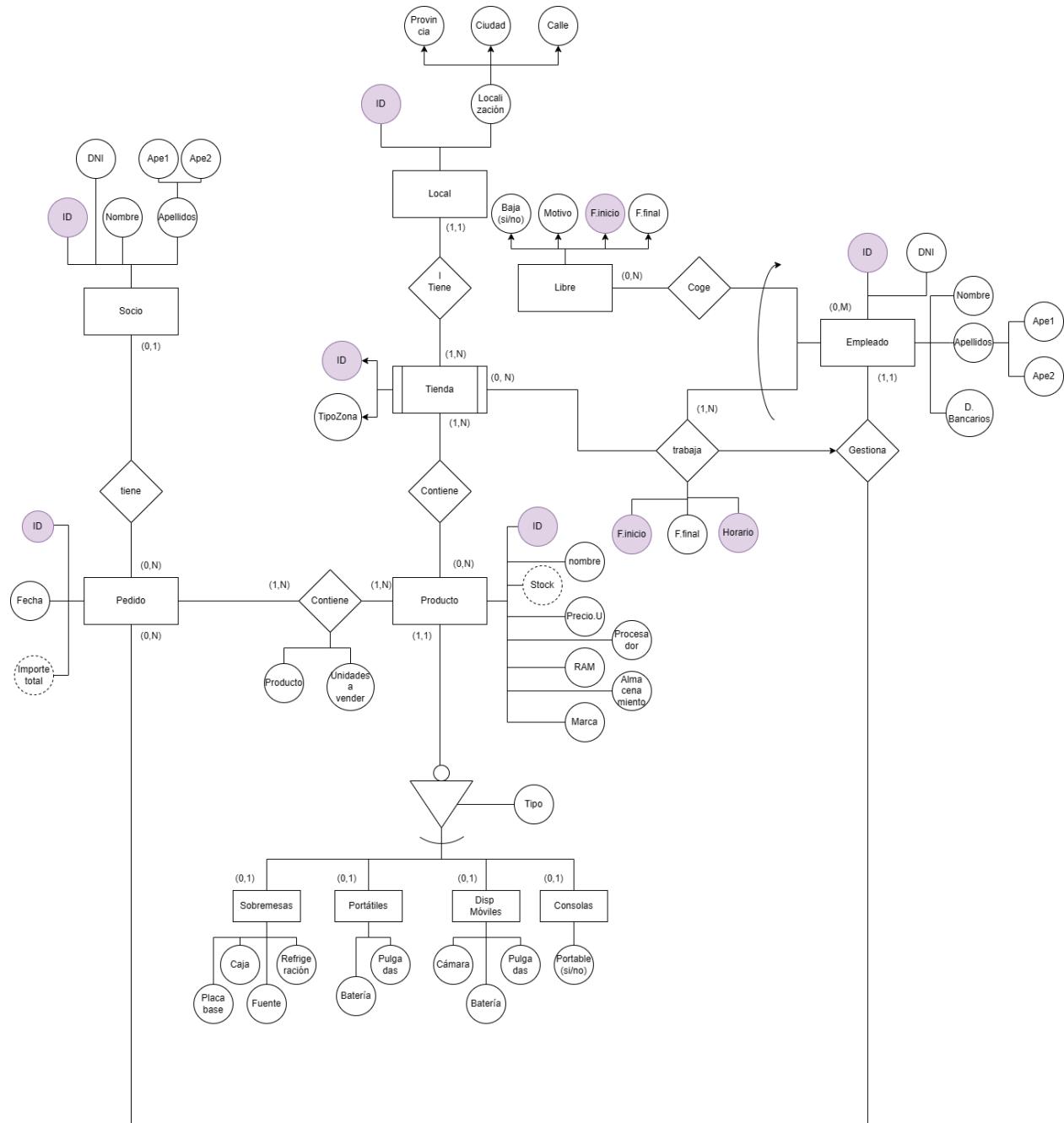
- La base de datos debe permitir registrar información detallada de los productos, incluyendo su tipo (sobremesa, portátil, consola, dispositivos móviles, etc.) y las características específicas de cada uno.
- Los productos deben asociarse con una o más zonas de distribución.
- Debe ser posible registrar pedidos realizados por socios.
- Cada pedido puede incluir múltiples productos a través de la tabla pedido_producto.
- Los pedidos deben vincularse a socios y locales específicos.
- Gestión de empleados y su relación con locales
- Los empleados deben estar asociados a uno o más locales mediante la tabla trabaja.
- Debe registrarse información básica de los empleados, como nombre, identificación y puesto.
- Debe almacenarse información sobre socios, incluyendo datos personales y su relación con los pedidos realizados.
- Los locales deben estar relacionados con una zona específica para facilitar la logística de distribución.
- Cada zona debe permitir asociar productos específicos a través de la tabla zona_producto.



- La base de datos debe permitir realizar consultas que relacionen socios, pedidos, productos y zonas.
- Debe garantizarse la integridad referencial entre tablas mediante claves primarias y foráneas.
- Los datos deben ser validados mediante restricciones (CHECK, NOT NULL, etc.), por ejemplo:
- Los precios y cantidades deben ser valores positivos.
- Los empleados deben estar vinculados a al menos un local.
- El sistema debe permitir restringir acceso para operaciones sensibles como eliminación o modificación de datos, pudiendo hacer estas operaciones pero de manera controlada



Diagrama entidad-relación

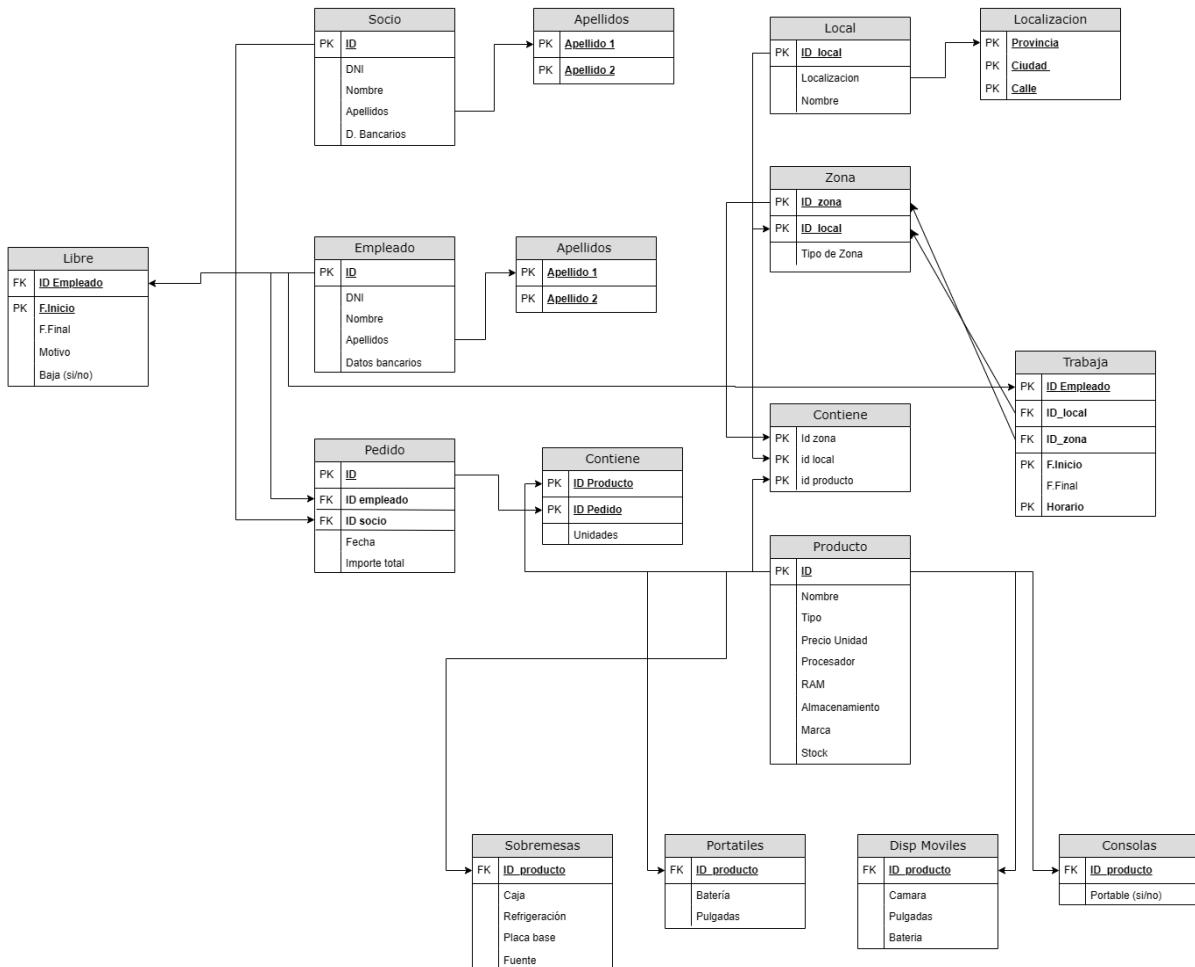


Restricciones semánticas

- Un empleado que no esté trabajando no puede coger la baja/vacaciones
- Las franjas de fechas de la tabla trabaja no se pueden solapar con las franjas de fechas de la tabla libre, siempre y cuando sea el mismo empleado en ambas
- Un trabajador solo puede gestionar un pedido si y sólo si está trabajando



Diagrama relacional





Base de datos SQL

Tablas base y tipos

Tipos

Tipo_producto

Python

```
CREATE TYPE TIPO_PRODUCTO AS ENUM('sobremesa', 'portatil', 'dispositivos moviles', 'consolas');
```

Tipo_zona

Python

```
CREATE TYPE TIPO_ZONA AS ENUM('almacen', 'tienda', 'reparacion');
```

Local

Python

```
CREATE TABLE LOCAL (
    id_local INT,
    provincia VARCHAR(100) NOT NULL CHECK (provincia <> ''),
    ciudad VARCHAR(100) NOT NULL CHECK (ciudad <> ''),
    calle VARCHAR(100) NOT NULL CHECK (calle <> ''),
    nombre VARCHAR(100) NOT NULL CHECK (nombre <> ''),
    PRIMARY KEY(id_local, provincia, ciudad, calle)
);
```

Esta es la tabla encargada de gestionar todas las tiendas. Guarda los datos que se esperan para gestionar varias tiendas, como su localización, un id y el nombre de la tienda (nombre del local)



Zona

Python

```
CREATE TABLE ZONA (
    id_zona INT NOT NULL,
    id_local INT NOT NULL,
    provincia VARCHAR(100) NOT NULL,
    ciudad VARCHAR(100) NOT NULL,
    calle VARCHAR(100) NOT NULL,
    tipo TIPO_ZONA NOT NULL,
    PRIMARY KEY(id_zona, id_local, provincia, ciudad, calle),
    FOREIGN KEY (id_local, provincia, ciudad, calle) REFERENCES LOCAL(id_local,
    provincia, ciudad, calle)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Esta la tabla zona, que como se puede observar, es una entidad débil, porque depende de la tabla tienda para poder identificarse de manera inequívoca.

Se hacen las verificaciones pertinentes dentro de la tabla, así como establecer el update y el delete en cascada

Producto

Python

```
CREATE TABLE PRODUCTO (
    id_producto INT,
    nombre VARCHAR(100) NOT NULL CHECK (nombre <> ''),
    tipo TIPO_PRODUCTO NOT NULL,
    precio_unidad DECIMAL(10, 2) CHECK (precio_unidad > 0),
    ram VARCHAR(100) NOT NULL CHECK (ram <> ''),
    almacenamiento VARCHAR(100) NOT NULL CHECK (almacenamiento <> ''),
    marca VARCHAR(100) NOT NULL CHECK (marca <> ''),
    stock INT CHECK (stock >= 0),
    PRIMARY KEY (id_producto)
);
```

Esta tabla almacena todos los productos disponibles, si un producto no aparece en esta tabla significa que no hay unidades disponibles. Hacemos las verificaciones pertinentes para mantener la integridad de los datos



Zona_producto

Python

```
CREATE TABLE ZONA_PRODUCTO (
    id_local INT NOT NULL,
    id_zona INT NOT NULL,
    id_producto INT UNIQUE NOT NULL,
    provincia VARCHAR(100) NOT NULL,
    ciudad VARCHAR(100) NOT NULL,
    calle VARCHAR(100) NOT NULL,
    PRIMARY KEY(id_local, id_zona, id_producto ,provincia, ciudad, calle),

    FOREIGN KEY (id_zona, id_local, provincia, ciudad, calle) REFERENCES
ZONA(id_zona, id_local, provincia, ciudad, calle) ON DELETE CASCADE ON UPDATE
CASCADE,
    FOREIGN KEY (id_producto) REFERENCES PRODUCTO(id_producto) ON DELETE CASCADE
ON UPDATE CASCADE
);
```

Esta tabla relaciona las zonas con el producto. De manera que nos permite manejar los productos que están en cada zona

Empleado

Python

```
CREATE TABLE EMPLEADO (
    id_empleado INT PRIMARY KEY CHECK (id_empleado > 0),
    dni VARCHAR(9) UNIQUE NOT NULL,
    CONSTRAINT dni_valido CHECK (dni ~ '^[0-9]{8}[A-Z]$'),
    nombre VARCHAR(100) NOT NULL CHECK (nombre <> ''),
    apellidos VARCHAR(100) NOT NULL CHECK (apellidos <> ''),
    datos_bancarios VARCHAR(255) NOT NULL CHECK (datos_bancarios <> '')
);
```

Se almacenan todos los datos necesarios de los empleados



Socio

Python

```
CREATE TABLE SOCIO (
    id_socio INT PRIMARY KEY,
    dni VARCHAR(9) UNIQUE NOT NULL,
    nombre VARCHAR(100) NOT NULL CHECK (nombre <> ''),
    apellidos VARCHAR(100) NOT NULL CHECK (apellidos <> ''),
    datos_bancarios VARCHAR(60) NOT NULL CHECK (datos_bancarios <> ''), -- 34
    para poner datos del estilo IBAN, de formato largo
    CONSTRAINT dni_valido CHECK (dni ~ '^[0-9]{8}[A-Z]$')
);
```

Se almacenan todos los datos necesarios para los socios. Así como un constraint para que el dni siga un formato específico

Pedido

Python

```
CREATE TABLE PEDIDO (
    id_pedido INT PRIMARY KEY,
    id_empleado INT,
    id_socio INT,
    fecha DATE NOT NULL CHECK (fecha <= CURRENT_DATE),
    importe_total DECIMAL(10, 2) NOT NULL CHECK (importe_total >= 0) DEFAULT 0,

    FOREIGN KEY (id_empleado) REFERENCES EMPLEADO(id_empleado) ON DELETE RESTRICT
    ON UPDATE RESTRICT,
    FOREIGN KEY (id_socio) REFERENCES SOCIO(id_socio) ON DELETE
    SET NULL ON UPDATE CASCADE
);
```

Tabla que maneja los distintos pedidos. Con una serie de consideraciones de seguridad respecto a las operaciones de delete y update. El atributo importe_total es un atributo calculado mediante el siguiente trigger:

Python

```
CREATE OR REPLACE FUNCTION calcular_importe_total()
RETURNS TRIGGER AS $$
BEGIN
```



```
-- Actualizar el importe total del pedido
UPDATE PEDIDO
SET importe_total = (
    SELECT SUM(pp.unidades * p.precio_unidad)
    FROM PEDIDO_PRODUCTO pp
    INNER JOIN PRODUCTO p ON pp.id_producto = p.id_producto
    WHERE pp.id_pedido = NEW.id_pedido
)
WHERE id_pedido = NEW.id_pedido;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Crear el trigger
CREATE TRIGGER trig_actualizar_importe_total
AFTER INSERT OR UPDATE ON PEDIDO_PRODUCTO
FOR EACH ROW
EXECUTE FUNCTION calcular_importe_total();
```

Este calcula la suma de todos los importes de los productos



Trabaja

Python

```
CREATE TABLE TRABAJA (
    id_empleado INT,
    id_local INT,
    id_zona INT,
    provincia VARCHAR(100) NOT NULL,
    ciudad VARCHAR(100) NOT NULL,
    calle VARCHAR(100) NOT NULL,
    fecha_inicio DATE,
    fecha_final DATE,
    horario TIME[],

    PRIMARY KEY(id_empleado,fecha_inicio, horario),
    FOREIGN KEY (id_empleado) REFERENCES EMPLEADO(id_empleado) ON DELETE CASCADE
ON UPDATE CASCADE,
    -- PRIMARY KEY(id_zona, id_local, provincia, ciudad, calle),
    -- Si se borra un empleado, se elimina su historial
    FOREIGN KEY (id_zona, id_local, provincia, ciudad, calle) REFERENCES
ZONA(id_zona, id_local, provincia, ciudad, calle) ON DELETE
CASCADE ON UPDATE CASCADE -- Si se cambia el id de Zona, se actualiza en
Trabaja
);
```

Esta tabla relaciona los empleados con la tienda y zona en la que trabajan. Con consideraciones a la hora de hacer ciertas operaciones, como la de update. Que si se cambia el id de la zona, esta también se cambia en esta tabla

Pedido_producto

Python

```
CREATE TABLE PEDIDO_PRODUCTO (
    id_producto INT,
    id_pedido INT,
    unidades INT CHECK (unidades > 0),
    PRIMARY KEY(id_producto, id_pedido),
    -- Llave compuesta para evitar duplicidad
    FOREIGN KEY (id_producto) REFERENCES PRODUCTO(id_producto) ON DELETE CASCADE
ON UPDATE CASCADE,
    -- Si se cambia el id del producto, se actualiza en Contiene
```



```
FOREIGN KEY (id_pedido) REFERENCES PEDIDO(id_pedido) ON DELETE CASCADE ON
UPDATE CASCADE -- Si se cambia el id del pedido, se actualiza en Contiene
);
```

Tabla pedido_producto, que relaciona los pedidos con los productos. Cada pedido contiene una serie de productos que hacen referencia a la tabla productos.

Libre

Python

```
CREATE TABLE LIBRE (
    id_empleado INT NOT NULL,
    fecha_inicio DATE NOT NULL,
    fecha_final DATE NOT NULL CHECK (fecha_final > fecha_inicio),
    motivo VARCHAR(100) NOT NULL CHECK (motivo <> ''),
    baja BOOLEAN NOT NULL,

    PRIMARY KEY(fecha_inicio),

    FOREIGN KEY(id_empleado) REFERENCES EMPLEADO(id_empleado)

);
```

Tabla libre, en esta tabla se especifica la franja de fechas que tiene un empleado libres, ya sea por baja o por vacaciones. Se toma en cuenta la opción de registrar el motivo de la baja.

Herencia

Todas las siguientes tablas heredan de producto, mediante el atributo tipo

Sobremesa

Python

```
-- Table sobremesa
CREATE TABLE SOBREMESA (
    id_producto INT NOT NULL,
    caja VARCHAR(100) NOT NULL CHECK (caja <> ''),
    refrigeracion VARCHAR(100) NOT NULL CHECK (refrigeracion <> ''),
```



```
placa_base VARCHAR(100) NOT NULL CHECK (placa_base <> ''),
fuente VARCHAR(100) NOT NULL CHECK (fuente <> ''),
PRIMARY KEY(id_producto),
FOREIGN KEY(id_producto) REFERENCES PRODUCTO(id_producto)
);
```

Portátiles

```
Python
CREATE TABLE PORTATILES(
    id_producto INT NOT NULL,
    bateria VARCHAR(100) NOT NULL CHECK (bateria <> ''),
    pulgadas VARCHAR(100) NOT NULL CHECK (pulgadas <> ''),
    PRIMARY KEY(id_producto),
    FOREIGN KEY(id_producto) REFERENCES PRODUCTO(id_producto)
);
```

Dispositivos_móviles

```
Python
CREATE TABLE DISPOSITIVOS_MOVILES(
    id_producto INT NOT NULL,
    bateria VARCHAR(100) NOT NULL CHECK (bateria <> ''),
    pulgadas VARCHAR(100) NOT NULL CHECK (pulgadas <> ''),
    camara VARCHAR(100) NOT NULL CHECK (camara <> ''),
    PRIMARY KEY(id_producto),
    FOREIGN KEY(id_producto) REFERENCES PRODUCTO(id_producto)
);
```



Consola

```
Python
CREATE TABLE CONSOLA(
    id_producto INT NOT NULL,
    portable BOOLEAN NOT NULL,
    PRIMARY KEY(id_producto),
    FOREIGN KEY(id_producto) REFERENCES PRODUCTO(id_producto)
);
```

Inclusividad

Empleado-trabaja

```
Python
CREATE OR REPLACE FUNCTION verificar_empleado_en_trabaja()
RETURNS TRIGGER AS $$$
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM TRABAJA
        WHERE id_empleado = NEW.id_empleado
    ) THEN
        RAISE EXCEPTION 'El empleado debe estar registrado en la tabla TRABAJA
para poder estar en LIBRE';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_verificar_empleado_en_trabaja
BEFORE INSERT ON LIBRE
FOR EACH ROW
EXECUTE FUNCTION verificar_empleado_en_trabaja();
```

Verificar que un empleado está en la tabla TRABAJA antes de insertarlo en LIBRE



Pedido-trabajo

Python

```
CREATE OR REPLACE FUNCTION verificar_empleado_pedido()
RETURNS TRIGGER AS $$

BEGIN
    -- Verificar si el empleado está en LIBRE durante la fecha del pedido
    IF EXISTS (
        SELECT 1
        FROM LIBRE
        WHERE id_empleado = NEW.id_empleado
        AND NEW.fecha BETWEEN fecha_inicio AND fecha_final
    ) THEN
        RAISE EXCEPTION 'El empleado está de vacaciones y no puede asignarse al
pedido';
    END IF;

    -- Verificar si el empleado está trabajando en TRABAJA durante la fecha del
pedido
    IF NOT EXISTS (
        SELECT 1
        FROM TRABAJA
        WHERE id_empleado = NEW.id_empleado
        AND NEW.fecha BETWEEN fecha_inicio AND COALESCE(fecha_final,
NEW.fecha)
    ) THEN
        RAISE EXCEPTION 'El empleado no está trabajando durante la fecha del
pedido';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_verificar_empleado_pedido
BEFORE INSERT ON PEDIDO
FOR EACH ROW
EXECUTE FUNCTION verificar_empleado_pedido();
```

Verificar que un empleado no está de vacaciones o fuera de horario al asignarle un pedido



Ejemplos de consultas

SELECT

INSERT

Local

```
-- Local
INSERT INTO LOCAL (id_local, provincia, ciudad, calle, nombre) VALUES
(1, 'Madrid', 'Madrid', 'Gran Vía', 'Store A'),
(2, 'Barcelona', 'Barcelona', 'Passeig de Gràcia', 'Store B'),
(3, 'Valencia', 'Valencia', 'Avenida del Oeste', 'Store C'),
(4, 'Sevilla', 'Sevilla', 'Calle Tetuán', 'Store D'),
(5, 'Madrid', 'Madrid', 'Calle de Preciados', 'Store E'),
(6, 'Bilbao', 'Bilbao', 'Gran Vía Don Diego López de Haro', 'Store F'),
(7, 'Zaragoza', 'Zaragoza', 'Calle del Coso', 'Store G'),
(8, 'Murcia', 'Murcia', 'Avenida de la Constitución', 'Store H'),
(9, 'Granada', 'Granada', 'Calle de los Reyes Católicos', 'Store I'),
(10, 'Palma', 'Palma de Mallorca', 'Carrer de Sant Miquel', 'Store J'),
(11, 'Málaga', 'Málaga', 'Calle Larios', 'Store K'),
(12, 'Alicante', 'Alicante', 'Calle Maisonnave', 'Store L'),
(13, 'Vigo', 'Vigo', 'Calle Urzáiz', 'Store M'),
(14, 'Gijón', 'Gijón', 'Calle de la Corrida', 'Store N'),
(15, 'Córdoba', 'Córdoba', 'Calle Capitulares', 'Store O'),
(16, 'Salamanca', 'Salamanca', 'Calle Toro', 'Store P'),
(17, 'Oviedo', 'Oviedo', 'Calle Uría', 'Store Q'),
(18, 'Santander', 'Santander', 'Calle Cádiz', 'Store R'),
(19, 'Valladolid', 'Valladolid', 'Calle de la Constitución', 'Store S'),
(20, 'Logroño', 'Logroño', 'Calle Portales', 'Store T');
INSERT 0 20
```

```
-- LOCAL
INSERT INTO LOCAL (id_local, provincia, ciudad, calle, nombre) VALUES (1, 'Madrid', 'Madrid', 'Gran Vía', '');
psql:bad_insert.sql:3: ERROR: new row for relation "local" violates check constraint "local_nombre_check"
DETAIL: Failing row contains (1, Madrid, Madrid, Gran Vía, ).
```



Zona

```
-- Zona
INSERT INTO ZONA (id_zona, id_local, provincia, ciudad, calle, tipo) VALUES
(1, 1, 'Madrid', 'Madrid', 'Gran Vía', 'tienda'),
(2, 2, 'Barcelona', 'Barcelona', 'Passeig de Gràcia', 'almacen'),
(3, 3, 'Valencia', 'Valencia', 'Avenida del Oeste', 'reparacion'),
(4, 4, 'Sevilla', 'Sevilla', 'Calle Tetuán', 'tienda'),
(5, 5, 'Madrid', 'Madrid', 'Calle de Preciados', 'almacen'),
(6, 6, 'Bilbao', 'Bilbao', 'Gran Vía Don Diego López de Haro', 'reparacion'),
(7, 7, 'Zaragoza', 'Zaragoza', 'Calle del Coso', 'tienda'),
(8, 8, 'Murcia', 'Murcia', 'Avenida de la Constitución', 'almacen'),
(9, 9, 'Granada', 'Granada', 'Calle de los Reyes Católicos', 'reparacion'),
(10, 10, 'Palma', 'Palma de Mallorca', 'Carrer de Sant Miquel', 'tienda'),
(11, 11, 'Málaga', 'Málaga', 'Calle Larios', 'almacen'),
(12, 12, 'Alicante', 'Alicante', 'Calle Maisonnave', 'reparacion'),
(13, 13, 'Vigo', 'Vigo', 'Calle Urzáiz', 'tienda'),
(14, 14, 'Gijón', 'Gijón', 'Calle de la Corrida', 'almacen'),
(15, 15, 'Córdoba', 'Córdoba', 'Calle Capitulares', 'reparacion'),
(16, 16, 'Salamanca', 'Salamanca', 'Calle Toro', 'tienda'),
(17, 17, 'Oviedo', 'Oviedo', 'Calle Uría', 'almacen'),
(18, 18, 'Santander', 'Santander', 'Calle Cádiz', 'reparacion'),
(19, 19, 'Valladolid', 'Valladolid', 'Calle de la Constitución', 'tienda'),
(20, 20, 'Logroño', 'Logroño', 'Calle Portales', 'almacen');
INSERT 0 20
```

```
-- Zona
INSERT INTO ZONA (id_zona, id_local, provincia, ciudad, calle, tipo) VALUES
(1, 1, 'Madrid', 'Madrid', 'Gran Vía', ''); -- Invalid tipo (empty string)
psql:bad_insert.sql:27: ERROR: invalid input value for enum tipo_zona: ""
LINE 2: (1, 1, 'Madrid', 'Madrid', 'Gran Vía', '');
```



Producto

```
-- Producto
INSERT INTO PRODUCTO (id_producto, nombre, tipo, precio_unidad, ram, almacenamiento, marca, stock) VALUES
(1, 'Laptop A', 'sobremesa', 1200.50, '8 GB', '512 GB SSD', 'HP', 100),
(2, 'Laptop B', 'portatil', 1500.00, '16 GB', '1 TB SSD', 'Dell', 50),
(3, 'Smartphone X', 'dispositivos moviles', 700.99, '6 GB', '128 GB', 'Samsung', 200),
(4, 'Smartphone Y', 'dispositivos moviles', 800.00, '8 GB', '256 GB', 'Apple', 150),
(5, 'Consola Play X', 'consolas', 400.00, '4 GB', '1 TB', 'Sony', 300),
(6, 'Consola Xbox Z', 'consolas', 450.00, '8 GB', '1 TB', 'Microsoft', 180),
(7, 'Desktop Tower', 'sobremesa', 1000.00, '16 GB', '2 TB HDD', 'Lenovo', 120),
(8, 'Tablet A', 'portatil', 350.00, '4 GB', '64 GB', 'Huawei', 250),
(9, 'Tablet B', 'portatil', 400.00, '6 GB', '128 GB', 'Samsung', 180),
(10, 'Smartwatch C', 'dispositivos moviles', 250.00, '2 GB', '16 GB', 'Garmin', 500),
(11, 'Laptop C', 'portatil', 1300.00, '8 GB', '256 GB SSD', 'Acer', 110),
(12, 'Smartphone Z', 'dispositivos moviles', 900.00, '8 GB', '512 GB', 'Google', 220),
(13, 'Desktop Pro', 'sobremesa', 1700.00, '32 GB', '2 TB SSD', 'Apple', 90),
(14, 'Laptop X', 'portatil', 2000.00, '32 GB', '1 TB SSD', 'MSI', 80),
(15, 'Smartphone V', 'dispositivos moviles', 750.00, '8 GB', '128 GB', 'OnePlus', 200),
(16, 'Smartphone W', 'dispositivos moviles', 850.00, '12 GB', '256 GB', 'Xiaomi', 160),
(17, 'Consola Switch', 'consolas', 350.00, '4 GB', '32 GB', 'Nintendo', 300),
(18, 'Smartwatch D', 'dispositivos moviles', 150.00, '1 GB', '8 GB', 'Fitbit', 600),
(19, 'Desktop Elite', 'sobremesa', 2500.00, '64 GB', '4 TB SSD', 'Alienware', 50),
(20, 'Laptop Y', 'portatil', 2200.00, '16 GB', '1 TB SSD', 'Razer', 40);
INSERT 0 20
```

```
-- Producto
INSERT INTO PRODUCTO (id_producto, nombre, tipo, precio_unidad, ram, almacenamiento, marca, stock) VALUES
(1, '', 'sobremesa', 1200.50, '8 GB', '512 GB SSD', 'HP', 100); -- Invalid nombre (empty string)
psql:bad_insert.sql:69: ERROR: new row for relation "producto" violates check constraint "producto_nombre_check"
DETAIL: Failing row contains (1, , sobremesa, 1200.50, 8 GB, 512 GB SSD, HP, 100).
```



Zona_producto

```
-- Zona_producto
INSERT INTO ZONA_PRODUCTO (id_local, id_zona, id_producto, provincia, ciudad, calle) VALUES
(1, 1, 1, 'Madrid', 'Madrid', 'Gran Vía'),
(2, 2, 2, 'Barcelona', 'Barcelona', 'Passeig de Gràcia'),
(3, 3, 3, 'Valencia', 'Valencia', 'Avenida del Oeste'),
(4, 4, 4, 'Sevilla', 'Sevilla', 'Calle Tetuán'),
(5, 5, 5, 'Madrid', 'Madrid', 'Calle de Preciados'),
(6, 6, 6, 'Bilbao', 'Bilbao', 'Gran Vía Don Diego López de Haro'),
(7, 7, 7, 'Zaragoza', 'Zaragoza', 'Calle del Coso'),
(8, 8, 8, 'Murcia', 'Murcia', 'Avenida de la Constitución'),
(9, 9, 9, 'Granada', 'Granada', 'Calle de los Reyes Católicos'),
(10, 10, 10, 'Palma', 'Palma de Mallorca', 'Carrer de Sant Miquel'),
(11, 11, 11, 'Málaga', 'Málaga', 'Calle Larios'),
(12, 12, 12, 'Alicante', 'Alicante', 'Calle Maisonnave'),
(13, 13, 13, 'Vigo', 'Vigo', 'Calle Urzáiz'),
(14, 14, 14, 'Gijón', 'Gijón', 'Calle de la Corrida'),
(15, 15, 15, 'Córdoba', 'Córdoba', 'Calle Capitulares'),
(16, 16, 16, 'Salamanca', 'Salamanca', 'Calle Toro'),
(17, 17, 17, 'Oviedo', 'Oviedo', 'Calle Uría'),
(18, 18, 18, 'Santander', 'Santander', 'Calle Cádiz'),
(19, 19, 19, 'Valladolid', 'Valladolid', 'Calle de la Constitución'),
(20, 20, 20, 'Logroño', 'Logroño', 'Calle Portales');
INSERT 0 20
```

```
-- Zona_producto
INSERT INTO ZONA_PRODUCTO (id_local, id_zona, id_producto, provincia, ciudad, calle) VALUES
(1, 1, 100, 'Madrid', 'Madrid', 'Gran Vía'); -- Invalid id_producto (does not exist in PRODUCTO)
psql:bad_insert.sql:109: ERROR: insert or update on table "zona_producto" violates foreign key constraint "zona_producto_id_producto_fkey"
DETAIL: Key (id_producto)=(100) is not present in table "producto".
```



Empleado

```
-- Empleado
INSERT INTO EMPLEADO (id_empleado, dni, nombre, apellidos, datos_bancarios) VALUES
(1, '12345678A', 'John', 'Doe', 'IBAN123456789'),
(2, '23456789B', 'Jane', 'Smith', 'IBAN987654321'),
(3, '34567890C', 'Carlos', 'Gomez', 'IBAN456123789'),
(4, '45678901D', 'Maria', 'Garcia', 'IBAN789123456'),
(5, '56789012E', 'Luis', 'Martinez', 'IBAN321456987'),
(6, '67890123F', 'Anna', 'Lopez', 'IBAN654321654'),
(7, '78901234G', 'Pedro', 'Hernandez', 'IBAN852963741'),
(8, '89012345H', 'Laura', 'Sanchez', 'IBAN741852963'),
(9, '90123456I', 'Marta', 'Perez', 'IBAN963741852'),
(10, '12345678J', 'David', 'Diaz', 'IBAN741963852'),
(11, '23456789K', 'Sara', 'Fernandez', 'IBAN258369741'),
(12, '34567890L', 'Antonio', 'Ruiz', 'IBAN369258147'),
(13, '45678901M', 'Eva', 'Ramirez', 'IBAN123987654'),
(14, '56789012N', 'Miguel', 'Jimenez', 'IBAN987321654'),
(15, '67890123O', 'Jose', 'Vazquez', 'IBAN654987321'),
(16, '78901234P', 'Elena', 'Gonzalez', 'IBAN258741963'),
(17, '89012345Q', 'Rosa', 'Mendez', 'IBAN369852741'),
(18, '90123456R', 'Pablo', 'Torres', 'IBAN852741963'),
(19, '12345678S', 'Isabel', 'Castro', 'IBAN147258369'),
(20, '23456789T', 'Victor', 'Moreno', 'IBAN258369147');
INSERT 0 20
```

```
-- Empleados
INSERT INTO EMPLEADO (dni, nombre, apellidos, datos_bancarios) VALUES
('12345A', 'John', 'Doe', 'IBAN123456789'), -- DNI is too short
('2345678B', 'Jane', 'Smith', 'IBAN987654321'), -- DNI has incorrect length
('34567890Z', '', 'Gomez', 'IBAN456123789'), -- Empty name
('45678901D', 'Maria', '', 'IBAN789123456'), -- Empty surname
('56789012E', 'Luis', 'Martinez', ''), -- Empty bank data
('67890123F', 'Anna', 'Lopez', ''); -- Empty bank data
psql:bad_insert.sql:152: ERROR: null value in column "id_empleado" of relation "empleado" violates not-null constraint
```



Socio

```
-- Socio
INSERT INTO SOCIO (id_socio, dni, nombre, apellidos, datos_bancarios) VALUES
(1, '12345678A', 'John', 'Doe', 'ES1234567890123456789012345678901234'),
(2, '23456789B', 'Jane', 'Smith', 'ES9876543210987654321098765432109876'),
(3, '34567890C', 'Carlos', 'Gomez', 'ES4561237890123456789012345678901234'),
(4, '45678901D', 'Maria', 'Garcia', 'ES7891234560987654321098765432109876'),
(5, '56789012E', 'Luis', 'Martinez', 'ES3214569870123456789012345678901234'),
(6, '67890123F', 'Anna', 'Lopez', 'ES6543216540987654321098765432109876'),
(7, '78901234G', 'Pedro', 'Hernandez', 'ES8529637410987654321098765432109876'),
(8, '89012345H', 'Laura', 'Sanchez', 'ES7418529630123456789012345678901234'),
(9, '90123456I', 'Marta', 'Perez', 'ES9637418520987654321098765432109876'),
(10, '12345678J', 'David', 'Diaz', 'ES7419638520123456789012345678901234'),
(11, '23456789K', 'Sara', 'Fernandez', 'ES2583697410987654321098765432109876'),
(12, '34567890L', 'Antonio', 'Ruiz', 'ES3692581470123456789012345678901234'),
(13, '45678901M', 'Eva', 'Ramirez', 'ES1239876540987654321098765432109876'),
(14, '56789012N', 'Miguel', 'Jimenez', 'ES9873216540123456789012345678901234'),
(15, '67890123O', 'Jose', 'Vazquez', 'ES6549873210987654321098765432109876'),
(16, '78901234P', 'Elena', 'Gonzalez', 'ES2587419630123456789012345678901234'),
(17, '89012345Q', 'Rosa', 'Mendez', 'ES3698527410987654321098765432109876'),
(18, '90123456R', 'Pablo', 'Torres', 'ES8527419630123456789012345678901234'),
(19, '12345678S', 'Isabel', 'Castro', 'ES1472583690987654321098765432109876'),
(20, '23456789T', 'Victor', 'Moreno', 'ES2583691470123456789012345678901234');
INSERT 0 20
```

```
-- Socio
INSERT INTO SOCIO (id_socio, dni, nombre, apellidos, datos_bancarios) VALUES
(1, '12345A', 'John', 'Doe', 'ES1234567890123456789012345678901234');
psql:bad_insert.sql:170: ERROR: new row for relation "socio" violates check constraint "dni_valido"
DETAIL: Failing row contains (1, 12345A, John, Doe, ES123456789012345678901234).
```

Trabaja

```
-- Trabaja
INSERT INTO TRABAJA (id_empleado, id_local, id_zona, provincia, ciudad, calle, fecha_inicio, fecha_final, horario) VALUES
(1, 1, 1, 'Madrid', 'Madrid', 'Gran Vía', '2023-01-01', NULL, ARRAY['09:00'::TIME, '17:00'::TIME]),
(2, 2, 2, 'Barcelona', 'Barcelona', 'Passeig de Gràcia', '2023-02-01', '2023-12-31', ARRAY['08:00'::TIME, '16:00'::TIME]),
(3, 3, 3, 'Valencia', 'Valencia', 'Avenida del Oeste', '2023-03-01', NULL, ARRAY['10:00'::TIME, '18:00'::TIME]),
(4, 4, 4, 'Sevilla', 'Sevilla', 'Calle Tetuán', '2023-04-01', '2023-10-31', ARRAY['07:00'::TIME, '15:00'::TIME]),
(5, 5, 5, 'Madrid', 'Madrid', 'Calle de Preciados', '2023-05-01', NULL, ARRAY['12:00'::TIME, '20:00'::TIME]),
(6, 6, 6, 'Bilbao', 'Bilbao', 'Gran Vía Don Diego López de Haro', '2023-06-01', '2023-12-31', ARRAY['09:00'::TIME, '17:00'::TIME]),
(7, 7, 7, 'Zaragoza', 'Zaragoza', 'Calle del Coso', '2023-07-01', NULL, ARRAY['08:00'::TIME, '16:00'::TIME]),
(8, 8, 8, 'Murcia', 'Murcia', 'Avenida de la Constitución', '2023-08-01', '2023-10-31', ARRAY['10:00'::TIME, '18:00'::TIME]),
(9, 9, 9, 'Granada', 'Granada', 'Calle de los Reyes Católicos', '2023-09-01', NULL, ARRAY['07:00'::TIME, '15:00'::TIME]),
(10, 10, 10, 'Palma', 'Palma de Mallorca', 'Carrer de Sant Miquel', '2023-10-01', '2023-12-31', ARRAY['12:00'::TIME, '20:00'::TIME]),
(11, 11, 11, 'Málaga', 'Málaga', 'Calle Larios', '2023-11-01', NULL, ARRAY['09:00'::TIME, '17:00'::TIME]),
(12, 12, 12, 'Alicante', 'Alicante', 'Calle Maisonnave', '2023-12-01', '2023-12-31', ARRAY['08:00'::TIME, '16:00'::TIME]),
(13, 13, 13, 'Vigo', 'Vigo', 'Calle Urzáiz', '2024-01-01', NULL, ARRAY['10:00'::TIME, '18:00'::TIME]),
(14, 14, 14, 'Gijón', 'Gijón', 'Calle de la Corrida', '2024-02-01', '2024-12-31', ARRAY['07:00'::TIME, '15:00'::TIME]),
(15, 15, 15, 'Córdoba', 'Córdoba', 'Calle Capitulares', '2024-03-01', NULL, ARRAY['12:00'::TIME, '20:00'::TIME]),
(16, 16, 16, 'Salamanca', 'Salamanca', 'Calle Toro', '2024-04-01', '2024-10-31', ARRAY['09:00'::TIME, '17:00'::TIME]),
(17, 17, 17, 'Oviedo', 'Oviedo', 'Calle Uria', '2024-05-01', NULL, ARRAY['08:00'::TIME, '16:00'::TIME]),
(18, 18, 18, 'Santander', 'Santander', 'Calle Cádiz', '2024-06-01', '2024-12-31', ARRAY['10:00'::TIME, '18:00'::TIME]),
(19, 19, 19, 'Valladolid', 'Valladolid', 'Calle de la Constitución', '2024-07-01', NULL, ARRAY['07:00'::TIME, '15:00'::TIME]),
(20, 20, 20, 'Logroño', 'Logroño', 'Calle Portales', '2024-08-01', '2024-10-31', ARRAY['12:00'::TIME, '20:00'::TIME]);
INSERT 0 20
```

```
-- Trabaja
INSERT INTO TRABAJA (id_empleado, id_local, id_zona, provincia, ciudad, calle, fecha_inicio, fecha_final, horario) VALUES
(999, 1, 1, 'Madrid', 'Madrid', 'Gran Vía', '2024-01-01', '2024-12-31', ARRAY['09:00'::TIME, '17:00'::TIME]), -- id empleado no existe
(1, 1, 1, 'Madrid', 'Madrid', 'Gran Vía', '2024-12-31', '2024-12-31', ARRAY['09:00'::TIME, '17:00'::TIME]), -- fecha_final no puede ser igual o menor a fecha
(1, 999, 999, 'Madrid', 'Madrid', 'Gran Vía', '2024-01-01', '2024-12-31', ARRAY['09:00'::TIME, '17:00'::TIME]), -- id_zona/id_local no existe
(1, 1, 1, 'Madrid', 'Madrid', 'Gran Vía', '2024-01-01', '2024-12-31', ARRAY['09:00'::TIME, '17:00']), -- sin ::TIME
psql:bad_insert.sql:216: ERROR: column "horario" is of type time without time zone[] but expression is of type text[]
LINE 5: ...'Madrid', 'Gran Vía', '2024-01-01', '2024-12-31', ARRAY['09:...
^
HINT: You will need to rewrite or cast the expression.
```



Pedido

```
-- Pedido
INSERT INTO PEDIDO (id_pedido, id_empleado, id_socio, fecha, importe_total) VALUES
(1, 1, 1, '2024-12-01', 500.00),
(2, 1, 2, '2024-12-02', 1200.50),
(3, 3, NULL, '2024-12-05', 800.00), -- Pedido sin socio
(4, 3, 4, '2024-12-07', 450.75),
(5, 5, 5, '2024-12-10', 300.00),
(6, 5, NULL, '2024-12-12', 900.99), -- Pedido de un cliente no socio
(7, 7, 7, '2024-12-15', 1100.00),
(8, 7, 8, '2024-12-16', 750.25),
(9, 11, 9, '2024-12-17', 2000.00),
(10, 11, 10, '2024-11-20', 350.00),
(11, 13, 11, '2024-11-22', 400.00),
(12, 13, 12, '2024-11-25', 800.00),
(13, 15, 13, '2024-11-27', 450.00),
(14, 15, 14, '2024-11-27', 600.00),
(15, 17, 15, '2024-08-01', 700.00),
(16, 17, 16, '2024-08-02', 350.00),
(17, 19, 17, '2024-08-05', 400.00),
(18, 19, 18, '2024-08-07', 800.00),
(19, 9, 19, '2024-08-10', 450.00),
(20, 9, 20, '2024-08-12', 600.00);
INSERT 0 20
```

```
-- Pedido
INSERT INTO PEDIDO (id_pedido, id_empleado, id_socio, fecha, importe_total) VALUES
(1, 1, 1, '2024-12-17', 500.00),
(1, 2, 2, '2024-12-18', 600.00), -- id_pedido repetido
(2, 999, 1, '2024-12-17', 400.00), -- id_empleado no existe
(3, 3, NULL, '2024-12-17', 300.00),
(4, 4, 4, '2024-12-17', -200.00), -- importe_total negativo
(5, 5, 5, '2050-01-10', 100.00);
psql:bad_insert.sql:209: ERROR: duplicate key value violates unique constraint "pedido_pkey"
DETAIL: Key (id_pedido)=(1) already exists.
```



Pedido_producto

```
-- Pedido_producto
INSERT INTO PEDIDO_PRODUCTO (id_producto, id_pedido, unidades) VALUES
(1, 1, 2),
(2, 1, 1),
(3, 1, 4),
(4, 2, 3),
(5, 2, 2),
(6, 3, 5),
(7, 4, 1),
(8, 4, 3),
(9, 4, 2),
(10, 5, 1),
(11, 6, 3),
(12, 7, 2),
(13, 7, 1),
(14, 7, 3),
(15, 8, 4),
(16, 9, 1),
(17, 10, 2),
(18, 10, 1),
(19, 10, 3),
(20, 11, 4);
INSERT 0 20
```

```
-- Pedido
INSERT INTO PEDIDO (id_pedido, id_empleado, id_socio, fecha, importe_total) VALUES
(1, 1, 1, '2024-12-17', 500.00),
(1, 2, 2, '2024-12-18', 600.00), -- id_pedido repetido
(2, 999, 1, '2024-12-17', 400.00), -- id_empleado no existe
(3, 3, NULL, '2024-12-17', 300.00),
(4, 4, 4, '2024-12-17', -200.00), -- importe_total negativo
(5, 5, 5, '2050-01-10', 100.00);
psql:bad_insert.sql:209: ERROR: duplicate key value violates unique constraint "pedido_pkey"
DETAIL: Key (id_pedido)=(1) already exists.
```



Libre

```
-- Libre
INSERT INTO LIBRE (id_empleado, fecha_inicio, fecha_final, motivo, baja) VALUES
(1, '2024-01-10', '2024-01-15', 'Vacaciones anuales', FALSE),
(2, '2024-02-05', '2024-02-10', 'Permiso médico', TRUE),
(3, '2024-03-01', '2024-03-03', 'Asuntos personales', FALSE),
(4, '2024-03-15', '2024-03-18', 'Baja por accidente', TRUE),
(5, '2024-04-01', '2024-04-05', 'Viaje familiar', FALSE),
(6, '2024-04-20', '2024-04-25', 'Cuidado de un familiar enfermo', FALSE),
(7, '2024-05-10', '2024-05-12', 'Asistencia a una boda', FALSE),
(8, '2024-06-01', '2024-06-10', 'Vacaciones en el extranjero', FALSE),
(9, '2024-07-01', '2024-07-07', 'Reposo médico', TRUE),
(10, '2024-08-15', '2024-08-20', 'Mudanza', FALSE),
(11, '2024-09-01', '2024-09-03', 'Permiso por examen académico', FALSE),
(12, '2024-10-05', '2024-10-10', 'Baja por enfermedad prolongada', TRUE),
(13, '2024-11-01', '2024-11-07', 'Viaje de emergencia', FALSE),
(14, '2024-11-15', '2024-11-20', 'Asistencia a una conferencia', FALSE),
(15, '2024-12-01', '2024-12-05', 'Recuperación de cirugía', TRUE),
(16, '2024-12-10', '2024-12-15', 'Cuidados postnatales', FALSE),
(17, '2024-12-13', '2024-12-15', 'Visita familiar', FALSE),
(18, '2024-12-14', '2024-12-17', 'Permiso por duelo', FALSE),
(19, '2024-12-15', '2024-12-20', 'Viaje de estudios', FALSE),
(20, '2024-12-16', '2024-12-22', 'Baja médica', TRUE);
INSERT 0 20
```

```
-- Libre
INSERT INTO LIBRE (id_empleado, fecha_inicio, fecha_final, motivo, baja) VALUES
(999, '2024-12-17', '2024-12-31', 'Descanso', TRUE), -- id_empleado no existe
(1, '2024-12-31', '2024-12-30', 'Vacaciones', FALSE); -- fecha_final no puede ser menor o igual a fecha_inicio
psql:bad_insert.sql:227: ERROR: El empleado debe estar registrado en la tabla TRABAJA para poder estar en LIBRE
CONTEXT: PL/pgSQL function verificar_empleado_en_trabajo() line 8 at RAISE
```

Sobremesa

```
-- Sobremesa
INSERT INTO SOBREMESA (id_producto, caja, refrigeracion, placa_base, fuente) VALUES
(1, 'Torre estándar', 'Activa', 'Intel B560', '650W'),
(7, 'Torre compacta', 'Pasiva', 'AMD X570', '750W'),
(13, 'Torre personalizada', 'Activa', 'Intel Z590', '850W'),
(19, 'Torre de alto rendimiento', 'Activa', 'Intel Z690', '1000W');
INSERT 0 4
```

```
-- Sobremesa
INSERT INTO SOBREMESA (id_producto, caja, refrigeracion, placa_base, fuente) VALUES
(999, 'Torre normal', 'Activa', 'Intel B460', '650W'), -- id_producto no existe
(1, ' ', 'Activa', 'Intel B560', '650W'), -- caja no puede estar vacío
(1, 'Torre normal', ' ', 'Intel B560', '650W'), -- refrigeración no puede estar vacío
(1, 'Torre normal', 'Activa', ' ', '650W'), -- placa_base no puede estar vacío
(1, 'Torre normal', 'Activa', 'Intel B560', ' '); -- fuente no puede estar vacío
psql:bad_insert.sql:235: ERROR: new row for relation "sobremesa" violates check constraint "sobremesa_caja_check"
DETAIL: Failing row contains (1, , Activa, Intel B560, 650W).
```



Portatiles

```
-- Portatiles
INSERT INTO PORTATILES (id_producto, bateria, pulgadas) VALUES
(2, 'Li-Ion 68Wh', '15.6'),
(8, 'Li-Po 4500mAh', '10.1'),
(9, 'Li-Ion 5000mAh', '10.5'),
(11, 'Li-Ion 48Wh', '14'),
(14, 'Li-Ion 90Wh', '17'),
(20, 'Li-Ion 75Wh', '15.6');
INSERT 0 6
```

```
-- Portatiles
INSERT INTO PORTATILES (id_producto, bateria, pulgadas) VALUES
(999, 'Li-Ion 72Wh', '15.6'), -- id_producto no existe
(2, '', '15.6'), -- bateria no puede estar vacío
(2, 'Li-Ion 72Wh', ''); -- pulgadas no puede estar vacío
psql:bad_insert.sql:241: ERROR: new row for relation "portatiles" violates check constraint "portatiles_bateria_check"
DETAIL: Failing row contains (2, , 15.6).
```

Dispositivos_móviles

```
-- Dispositivos móviles
INSERT INTO DISPOSITIVOS_MOVILES (id_producto, bateria, pulgadas, camara) VALUES
(3, 'Li-Ion 4000mAh', '6.5', 'Triple 12MP'),
(4, 'Li-Ion 3500mAh', '6.1', 'Dual 12MP'),
(10, 'Li-Ion 300mAh', '1.3', 'GPS'),
(12, 'Li-Ion 4500mAh', '6.3', 'Doble 48MP'),
(15, 'Li-Ion 3800mAh', '6.55', 'Cuádruple 64MP'),
(16, 'Li-Ion 4000mAh', '6.8', 'Triple 108MP'),
(18, 'Li-Ion 150mAh', '1.4', 'Mono 8MP');
INSERT 0 7
```

```
-- Dispositivos móviles
INSERT INTO DISPOSITIVOS_MOVILES (id_producto, bateria, pulgadas, camara) VALUES
(999, 'Li-Ion 4500mAh', '6.5', 'Triple 12MP'), -- id_producto no existe
(3, '', '6.5', 'Triple 12MP'), -- bateria no puede estar vacío
(3, 'Li-Ion 4000mAh', '', 'Triple 12MP'), -- pulgadas no puede estar vacío
(3, 'Li-Ion 4000mAh', '6.5', ''), -- camara no puede estar vacío
```

Consola

```
-- Consola
INSERT INTO CONSOLA (id_producto, portable) VALUES
(5, FALSE),
(6, FALSE),
(17, TRUE);
INSERT 0 3
```

```
-- Consola
INSERT INTO CONSOLA (id_producto, portable) VALUES
(999, TRUE), -- id_producto no existe
(5, NULL); -- portable debe ser BOOLEAN (TRUE o FALSE)
psql:bad_insert.sql:253: ERROR: syntax error at or near "INSERT"
LINE 7: INSERT INTO CONSOLA (id_producto, portable) VALUES
^
```



UPDATE

Local

```
-- Local
SELECT 'TABLA LOCAL';
?column?
-----
 TABLA LOCAL
(1 row)

UPDATE LOCAL SET provincia = 'Valencia', calle = 'Calle Nueva' WHERE id_local = 2;
UPDATE 1
UPDATE LOCAL SET nombre = 'Store Updated' WHERE id_local = 3;
UPDATE 1
UPDATE LOCAL SET ciudad = 'Sevilla' WHERE id_local = 4;
UPDATE 1
UPDATE LOCAL SET calle = 'Avenida Principal' WHERE id_local = 5;
UPDATE 1
```

```
-- Local
UPDATE LOCAL SET id_local = NULL WHERE id_local = 6;
psql:bad_update.sql:2: ERROR: null value in column "id_local" of relation "local" violates not-null constraint
DETAIL: Failing row contains (null, Bilbao, Bilbao, Gran Vía Don Diego López de Haro, Store F).
```

Zona

```
-- Zona
SELECT 'TABLA ZONA';
?column?
-----
 TABLA ZONA
(1 row)

UPDATE ZONA SET tipo = 'tienda' WHERE id_zona = 1;
UPDATE 1
UPDATE ZONA SET provincia = 'Bilbao' WHERE id_zona = 6;
UPDATE 1
UPDATE ZONA SET ciudad = 'Vigo' WHERE id_zona = 13;
UPDATE 1
UPDATE ZONA SET tipo = 'almacen' WHERE id_zona = 16;
UPDATE 1
```

```
UPDATE ZONA SET provincia = '' WHERE id_zona = 1; -- Fails due to empty string in 'provincia'
psql:bad_update.sql:8: ERROR: insert or update on table "zona" violates foreign key constraint "zona_id_local_provincia_ciudad_calle_fkey"
DETAIL: Key (id_local, provincia, ciudad, calle)=(1, , Madrid, Gran Vía) is not present in table "local".
```



Producto

```
-- Producto
SELECT 'TABLA PRODUCTO';
?column?
-----
TABLA PRODUCTO
(1 row)

UPDATE PRODUCTO SET stock = 120 WHERE id_producto = 1;
UPDATE 1
UPDATE PRODUCTO SET precio_unidad = 750.00 WHERE id_producto = 3;
UPDATE 1
UPDATE PRODUCTO SET almacenamiento = '2 TB' WHERE id_producto = 5;
UPDATE 1
UPDATE PRODUCTO SET ram = '16 GB' WHERE id_producto = 4;
UPDATE 1
UPDATE PRODUCTO SET marca = 'Lenovo' WHERE id_producto = 11;
UPDATE 1
```

```
UPDATE PRODUCTO SET precio = -100 WHERE id_producto = 5; -- Fails due to negative value in 'precio'
psql:bad_update.sql:15: ERROR: column "precio" of relation "producto" does not exist
LINE 1: UPDATE PRODUCTO SET precio = -100 WHERE id_producto = 5;
```

Zona_producto

```
-- Zona_producto
SELECT 'TABLA ZONA_PRODUCTO';
?column?
-----
TABLA ZONA_PRODUCTO
(1 row)

UPDATE ZONA_PRODUCTO SET provincia = 'Madrid', ciudad = 'Madrid', calle = 'Gran Vía' WHERE id_local = 1 AND id_zona = 1 AND id_producto = 1;
UPDATE 1
UPDATE ZONA_PRODUCTO SET provincia = 'Valencia', ciudad = 'Valencia', calle = 'Avenida del Oeste' WHERE id_local = 3 AND id_zona = 3 AND id_producto = 3;
UPDATE 1
UPDATE ZONA_PRODUCTO SET ciudad = 'Sevilla', calle = 'Calle Tetuán' WHERE id_local = 4 AND id_zona = 4 AND id_producto = 4;
UPDATE 1
UPDATE ZONA_PRODUCTO SET provincia = 'Bilbao', ciudad = 'Bilbao', calle = 'Gran Vía Don Diego López de Haro' WHERE id_local = 6 AND id_zona = 6 AND id_producto = 6;
UPDATE 1
```

```
UPDATE ZONA_PRODUCTO SET unidades = -5 WHERE id_zona = 3 AND id_producto = 4; -- Fails due to negative value in 'unidades'
psql:bad_update.sql:22: ERROR: column "unidades" of relation "zona_producto" does not exist
LINE 1: UPDATE ZONA_PRODUCTO SET unidades = -5 WHERE id_zona = 3 AND...
```



Empleado

```
-- Empleado
SELECT 'TABLA EMPLEADO';
?column?
-----
 TABLA EMPLEADO
(1 row)

UPDATE EMPLEADO SET nombre = 'Johnathan' WHERE id_empleado = 1;
UPDATE 1
UPDATE EMPLEADO SET apellidos = 'Johnson' WHERE id_empleado = 2;
UPDATE 1
UPDATE EMPLEADO SET dni = '34567890Z' WHERE id_empleado = 3;
UPDATE 1
UPDATE EMPLEADO SET datos_bancarios = 'IBAN654987654' WHERE id_empleado = 4;
UPDATE 1
UPDATE EMPLEADO SET nombre = 'Luis Miguel' WHERE id_empleado = 5;
UPDATE 1
```

```
UPDATE EMPLEADO SET fecha_nacimiento = NULL WHERE id_empleado = 2; -- Fails due to NULL in 'fecha_nacimiento'
psql:bad_update.sql:27: ERROR: column "fecha_nacimiento" of relation "empleado" does not exist
LINE 1: UPDATE EMPLEADO SET fecha_nacimiento = NULL WHERE id_emplad...
```

Socio

```
-- Socio
SELECT 'TABLA SOCIO';
?column?
-----
 TABLA SOCIO
(1 row)

UPDATE SOCIO SET nombre = 'Carlos Updated' WHERE id_socio = 1;
UPDATE 1
UPDATE SOCIO SET apellidos = 'Fernandez Updated' WHERE id_socio = 2;
UPDATE 1
UPDATE SOCIO SET datos_bancarios = 'ES76000000000000000000000000000002' WHERE id_socio = 3;
UPDATE 1
UPDATE SOCIO SET dni = '12345678Z' WHERE id_socio = 4;
UPDATE 1
UPDATE SOCIO SET nombre = 'Ana Updated' WHERE id_socio = 5;
UPDATE 1
```

```
-- Socio
UPDATE SOCIO SET dni = '1234' WHERE id_socio = 1;
psql:bad_update.sql:73: ERROR: new row for relation "socio" violates check constraint "dni_valido"
DETAIL: Failing row contains (1, 1234, Carlos Updated, Doe, ES1234567890123456789012345678901234).
```



Trabaja

```
-- Trabaja
SELECT 'TABLA TRABAJA';
?column?
-----
TABLA TRABAJA
(1 row)

UPDATE TRABAJA
SET fecha_final = '2024-12-31', horario = ARRAY['09:00'::TIME, '17:00'::TIME] WHERE id_empleado = 2 AND fecha_inicio = '2023-02-01';
UPDATE 1
UPDATE TRABAJA
SET fecha_final = '2023-10-31', horario = ARRAY['07:00'::TIME, '15:00'::TIME] WHERE id_empleado = 4 AND provincia = 'Sevilla';
UPDATE 1
UPDATE TRABAJA
SET fecha_inicio = '2024-05-01', horario = ARRAY['10:00'::TIME, '18:00'::TIME] WHERE id_empleado = 13 AND ciudad = 'Vigo';
UPDATE 1
```

```
UPDATE TRABAJA SET id_local = 999 WHERE id_empleado = 4; -- Fails due to no matching 'id_local' in the referenced table
psql:bad_update.sql:49: ERROR: insert or update on table "trabajas" violates foreign key constraint "trabajas_id_zona_id_local_provincia_ciudad_calle_fkey"
DETAIL: Key (id_zona, id_local, provincia, ciudad, calle)=(4, 999, Sevilla, Sevilla, Calle Tetuán) is not present in table "zona".
```

Pedido

```
-- Pedido
SELECT 'TABLA PEDIDO';
?column?
-----
TABLA PEDIDO
(1 row)

UPDATE PEDIDO SET importe_total = 550.00, fecha = '2024-12-03' WHERE id_pedido = 1;
UPDATE 1
UPDATE PEDIDO SET id_socio = 6, importe_total = 1300.00 WHERE id_pedido = 2;
UPDATE 1
UPDATE PEDIDO SET importe_total = 950.00 WHERE id_pedido = 6;
UPDATE 1
UPDATE PEDIDO SET id_socio = 10, fecha = '2024-12-18' WHERE id_pedido = 9;
UPDATE 1
UPDATE PEDIDO SET fecha = '2024-11-21', importe_total = 380.00 WHERE id_pedido = 10;
UPDATE 1
```

```
UPDATE PRODUCTO SET id_categoria = NULL WHERE id_producto = 2; -- Fails due to NULL value in foreign key 'id_categoria'
psql:bad_update.sql:16: ERROR: column "id_categoria" of relation "producto" does not exist
LINE 1: UPDATE PRODUCTO SET id_categoria = NULL WHERE id_producto = ...
```

Pedido_producto

```
-- PEDIDO_PRODUCTO
SELECT 'TABLA PEDIDO_PRODUCTO';
?column?
-----
TABLA PEDIDO_PRODUCTO
(1 row)

UPDATE PEDIDO_PRODUCTO SET unidades = 3 WHERE id_producto = 1 AND id_pedido = 1;
UPDATE 1
UPDATE PEDIDO_PRODUCTO SET unidades = 2 WHERE id_producto = 4 AND id_pedido = 2;
UPDATE 1
UPDATE PEDIDO_PRODUCTO SET unidades = 6 WHERE id_producto = 7 AND id_pedido = 4;
UPDATE 1
UPDATE PEDIDO_PRODUCTO SET unidades = 5 WHERE id_producto = 10 AND id_pedido = 5;
UPDATE 1
UPDATE PEDIDO_PRODUCTO SET unidades = 4 WHERE id_producto = 13 AND id_pedido = 7;
UPDATE 1
```

```
UPDATE PEDIDO_PRODUCTO SET unidades = -1 WHERE id_pedido = 1 AND id_producto = 2; -- Fails due to negative value in 'unidades'
psql:bad_update.sql:37: ERROR: new row for relation "pedido_producto" violates check constraint "pedido_producto_unidades_check"
DETAIL: Failing row contains (2, 1, -1).
```



Libre

```
-- Libre
SELECT 'TABLA LIBRE';
?column?
-----
TABLA LIBRE
(1 row)

UPDATE LIBRE SET fecha_final = '2024-01-18', motivo = 'Vacaciones anuales extendidas' WHERE id_empleado = 1 AND fecha_inicio = '2024-01-10';
UPDATE 1
UPDATE LIBRE SET fecha_inicio = '2024-03-02', fecha_final = '2024-03-04', motivo = 'Asuntos personales extendidos' WHERE id_empleado = 3 AND fecha_inicio = '2024-03-01';
UPDATE 1
UPDATE LIBRE SET motivo = 'Baja por accidente prolongada' WHERE id_empleado = 4 AND fecha_inicio = '2024-03-15';
UPDATE 1
UPDATE LIBRE SET fecha_final = '2024-10-12', motivo = 'Baja por enfermedad prolongada' WHERE id_empleado = 12 AND fecha_inicio = '2024-10-05';
UPDATE 1
UPDATE LIBRE SET baja = FALSE, motivo = 'Recuperación de cirugía exitosa' WHERE id_empleado = 15 AND fecha_inicio = '2024-12-01';
UPDATE 1
-- Sobremesa
SELECT 'TABLA SOBREMESA';
?column?
-----
TABLA SOBREMESA
(1 row)
```

```
UPDATE PEDIDO_PRODUCTO SET id_pedido = 5, id_producto = 999 WHERE id_pedido = 6; -- Fails due to no matching 'id_producto' in the referenced table
psql:bad_update.sql:41: ERROR: insert or update on table "pedido_producto" violates foreign key constraint "pedido_producto_id_producto_fkey"
DETAIL: Key (id_producto)=(999) is not present in table "producto".
```

Sobremesa

```
-- Sobremesa
SELECT 'TABLA SOBREMESA';
?column?
-----
TABLA SOBREMESA
(1 row)

UPDATE SOBREMESA SET caja = 'Torre avanzada', refrigeracion = 'Activa', placa_base = 'Intel Z590', fuente = '1000W' WHERE id_producto = 1;
UPDATE 1
UPDATE SOBREMESA SET caja = 'Torre compacta', refrigeracion = 'Pasiva', placa_base = 'AMD B550', fuente = '650W' WHERE id_producto = 7;
UPDATE 1
UPDATE SOBREMESA SET caja = 'Torre profesional', refrigeracion = 'Activa', placa_base = 'Intel Z690', fuente = '850W' WHERE id_producto = 13;
UPDATE 1
UPDATE SOBREMESA SET refrigeracion = 'Pasiva', fuente = '750W' WHERE id_producto = 19;
UPDATE 1
UPDATE SOBREMESA SET caja = 'Torre personalizada', placa_base = 'Intel B460' WHERE id_producto = 1;
UPDATE 1
```

```
UPDATE SOBREMESA SET caja = NULL WHERE id_producto = 1; -- Fails due to NULL in non-nullable field 'caja'
psql:bad_update.sql:52: ERROR: null value in column "caja" of relation "sobremesa" violates not-null constraint
DETAIL: Failing row contains (1, null, Activa, Intel B460, 1000W).
```

Portatiles

```
-- Portatiles
SELECT 'TABLA PORTATILES';
?column?
-----
TABLA PORTATILES
(1 row)

UPDATE PORTATILES SET bateria = 'Li-Ion 72Wh', pulgadas = '16"' WHERE id_producto = 2;
UPDATE 1
UPDATE PORTATILES SET bateria = 'Li-Po 5000mAh', pulgadas = '11.6"' WHERE id_producto = 8;
UPDATE 1
UPDATE PORTATILES SET bateria = 'Li-Ion 52Wh', pulgadas = '14"' WHERE id_producto = 9;
UPDATE 1
UPDATE PORTATILES SET bateria = 'Li-Ion 100Wh', pulgadas = '15.6"' WHERE id_producto = 14;
UPDATE 1
UPDATE PORTATILES SET pulgadas = '18.4"' WHERE id_producto = 20;
UPDATE 1
```

```
UPDATE PORTATILES SET pulgadas = '' WHERE id_producto = 8; -- Fails due to empty string in 'pulgadas'
psql:bad_update.sql:57: ERROR: new row for relation "portatiles" violates check constraint "portatiles_pulgadas_check"
DETAIL: Failing row contains (8, Li-Po 5000mAh, ).
```



Dispositivos_móviles

```
-- Dispositivos móviles
SELECT 'TABLA DISPOSITIVOS_MOVILES';
?column?
-----
TABLA DISPOSITIVOS_MOVILES
(1 row)

UPDATE DISPOSITIVOS_MOVILES SET bateria = 'Li-Ion 5000mAh', pulgadas = '6.7"', camara = 'Cuádruple 64MP' WHERE id_producto = 3;
UPDATE 1
UPDATE DISPOSITIVOS_MOVILES SET bateria = 'Li-Ion 4200mAh', camara = 'Dual 16MP' WHERE id_producto = 4;
UPDATE 1
UPDATE DISPOSITIVOS_MOVILES SET bateria = 'Li-Ion 5000mAh', camara = 'Triple 48MP' WHERE id_producto = 12;
UPDATE 1
UPDATE DISPOSITIVOS_MOVILES SET bateria = 'Li-Ion 4300mAh', camara = 'Cuádruple 108MP' WHERE id_producto = 15;
UPDATE 1
UPDATE DISPOSITIVOS_MOVILES SET camara = 'Mono 12MP', bateria = 'Li-Ion 200mAh' WHERE id_producto = 18;
UPDATE 1
```

```
UPDATE DISPOSITIVOS_MOVILES SET bateria = NULL WHERE id_producto = 3; -- Fails due to NULL in non-nullable field 'bateria'
psql:bad_update.sql:61: ERROR: null value in column "bateria" of relation "dispositivos_moviles" violates not-null constraint
DETAIL: Failing row contains (3, null, 6.7", Cuádruple 64MP).
```

Consola

```
-- Consola
SELECT 'TABLA CONSOLA';
?column?
-----
TABLA CONSOLA
(1 row)

UPDATE CONSOLA SET portable = TRUE WHERE id_producto = 5;
UPDATE 1
UPDATE CONSOLA SET portable = FALSE WHERE id_producto = 6;
UPDATE 1
UPDATE CONSOLA SET portable = TRUE WHERE id_producto = 17;
UPDATE 1
UPDATE CONSOLA SET portable = FALSE WHERE id_producto = 5;
UPDATE 1
UPDATE CONSOLA SET portable = TRUE WHERE id_producto = 6;
UPDATE 1
```

```
UPDATE CONSOLA SET id_producto = NULL WHERE id_producto = 5; -- Fails due to primary key being NULL
psql:bad_update.sql:69: ERROR: null value in column "id_producto" of relation "consola" violates not-null constraint
DETAIL: Failing row contains (null, f).
```

DELETE

Local

```
DELETE FROM local WHERE provincia = 'Madrid';
DELETE 2
```



Zona

```
DELETE FROM zona WHERE id_zona = 5;  
DELETE 0
```

Producto

```
DELETE FROM producto WHERE stock = 0;  
DELETE 0
```

Zona_producto

```
DELETE FROM zona_producto WHERE ciudad = 'Barcelona';  
DELETE 0
```

Empleado

```
DELETE FROM libre WHERE id_empleado = 2 AND fecha_inicio = '2024-01-01';  
DELETE 0
```

```
DELETE FROM empleado WHERE id_empleado NOT IN (SELECT id_empleado FROM trabaja);  
psql:good bad_delete.sql:9: ERROR: update or delete on table "empleado" violates foreign key constraint "pedido_id_empleado_fkey" on table "pedido"  
DETAIL: Key (id_empleado)=(7) is still referenced from table "pedido".
```

Socio

```
DELETE FROM socio WHERE id_socio NOT IN (SELECT id_socio FROM pedido WHERE id_socio IS NOT NULL);  
DELETE 0
```

Trabaja

```
DELETE FROM trabaja WHERE id_local = 10 AND fecha_final IS NOT NULL;  
DELETE 0
```

Pedido

```
DELETE FROM pedido WHERE id_pedido = 4;  
DELETE 0
```

Pedido_producto

```
DELETE FROM pedido_producto WHERE id_pedido = 3;  
DELETE 0
```



Libre

```
DELETE FROM libre WHERE id_empleado = 3;  
DELETE 0
```

Sobremesa

```
DELETE FROM sobremesa  
WHERE id_producto = 1;
```

Portatiles

```
DELETE FROM portatiles  
WHERE id_producto = 2;
```

Dispositivos_moviles

```
DELETE FROM dispositivos_moviles  
WHERE id_producto = 3;  
DELETE 1
```

Consola

```
DELETE FROM consola  
WHERE id_producto = 5;  
DELETE 1
```



Implementación de API

Operaciones CRUD

Estructuración de las rutas

En general la idea para definir las rutas ha seguido el mismo patrón, algunas básicas como local cuentan únicamente con una ruta base general sobre la que se han definido las operaciones CRUD, algunas otras como producto que tienen una ruta desde la raíz para realizar operaciones desde un punto de vista de administración centralizada y luego tienen una ruta que cuelga de “/local/<id_local>/zona/<id_zona>” y en esta se gestiona más a nivel específico, otras tablas siguen un patrón similar y luego hay casos excepcionales como Zona que al ser dependiente de local, su ruta solo tiene sentido que sea con prefijo la de local.

Cabe destacar que en la raíz de la api, al lanzarla hemos creado una guía o tabla de las rutas existentes en la api con una breve descripción de cada una.

Rutas Disponibles en la API

Producto

Método	Ruta	Descripción
GET	/producto	Lista todos los productos
GET	/producto/<id_producto>	Obtiene un producto por su id
POST	/producto	Crea un nuevo producto
PUT	/producto/<id_producto>	Actualiza un producto existente por su id
DELETE	/producto/<id_producto>	Elimina un producto por su id

Local

Método	Ruta	Descripción
GET	/local	Lista todos los locales
GET	/local/<id_local>	Obtiene un local por su id
POST	/local	Crea un nuevo local
PUT	/local/<id_local>	Actualiza un local existente
DELETE	/local/<id_local>	Elimina un local

Zona

Método	Ruta	Descripción
--------	------	-------------



Pruebas de API

```
^C(myenv) usuario@ubuntu:~/apiRest/postgresql_practice/ApiProyecto/books/flask_app$ flask --app app.py run
 * Serving Flask app 'app.py'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [18/Dec/2024 20:34:09] "PUT /local/2 HTTP/1.1" 200 -
127.0.0.1 - - [18/Dec/2024 20:34:14] "GET /local/2 HTTP/1.1" 200 -
127.0.0.1 - - [18/Dec/2024 20:35:12] "DELETE /local/2 HTTP/1.1" 200 -
127.0.0.1 - - [18/Dec/2024 20:48:13] "GET /local/2 HTTP/1.1" 404 -
127.0.0.1 - - [18/Dec/2024 20:48:22] "PUT /local/2 HTTP/1.1" 404 -
127.0.0.1 - - [18/Dec/2024 20:48:32] "POST /local/2 HTTP/1.1" 201 -
127.0.0.1 - - [18/Dec/2024 20:48:38] "PUT /local/2 HTTP/1.1" 200 -
```

En esta imagen se puede observar como, mediante el uso del framework flask. Podemos poner en funcionamiento nuestra API, obteniendo distintos códigos y cuerpo de respuesta según el resultado de la petición.

GET

GET por id

Python

```
@local_routes.route('/<int:id_local>', methods=['GET'])
def get_local_by_id(id_local):
    try:
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('SELECT * FROM LOCAL WHERE id_local = %s;', (id_local,))
        local = cur.fetchone()
        if local is None:
            return jsonify({"error": "Local no encontrado"}), 404
        local_data = {
            "id_local": local[0],
            "provincia": local[1],
            "ciudad": local[2],
            "calle": local[3],
            "nombre": local[4],
        }
        return jsonify(local_data), 200
    except Exception as e:
        return jsonify({"error": f"Error al obtener el local: {str(e)}"}), 500
    finally:
        if 'cur' in locals():
```



```
        cur.close()
if 'conn' in locals():
    conn.close()
```

GET http://127.0.0.1:5000/local/2

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

parameter

Status: 200 OK Size: 112 Bytes Time: 296 ms

```
1   {
2     "calle": "Passeig de Gràcia",
3     "ciudad": "Barcelona",
4     "id_local": 2,
5     "nombre": "Store B",
6     "provincia": "Barcelona"
7 }
```

GET general

```
Python
@local_routes.route('/', methods=[ 'GET' ])
def get_all_locales():
    try:
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('SELECT * FROM LOCAL ORDER BY id_local;')
        locales = cur.fetchall()
        locales_list = [
            {
                "id_local": l[0],
                "provincia": l[1],
                "ciudad": l[2],
                "calle": l[3],
```



```
        "nombre": l[4],
    }
    for l in locales
]
return jsonify(locales_list), 200
except Exception as e:
    return jsonify({"error": f"Error al obtener los locales: {str(e)}"}),
500
finally:
    if 'cur' in locals():
        cur.close()
    if 'conn' in locals():
        conn.close()
```

GET ▾ http://127.0.0.1:5000/local/

Query	Headers 2	Auth	Body	Tests	Pre Run
Status: 200 OK Size: 1.83 KB Time: 311 ms					
2	{				
3	"calle": "Passeig de Gràcia",				
4	"ciudad": "Barcelona",				
5	"id_local": 2,				
6	"nombre": "Store B Updated",				
7	"provincia": "Barcelona"				
8	},				
9	{				
10	"calle": "Avenida del Oeste",				
11	"ciudad": "Valencia",				
12	"id_local": 3,				
13	"nombre": "Store Updated",				
14	"provincia": "Valencia"				
15	},				
16	{				
17	"calle": "Calle Tetuán",				
18	"ciudad": "Sevilla",				
19	"id_local": 4,				
20	"nombre": "Store D",				
21	"provincia": "Sevilla"				
22	},				
23	{				
24	"calle": "Gran Vía Don Diego López de Haro",				
25	"ciudad": "Bilbao".				



POST

Python

```
@local_routes.route('/', methods=['POST'])
def create_local():
    data = request.json
    try:
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('''
            INSERT INTO LOCAL (id_local, provincia, ciudad, calle, nombre)
            VALUES (%s, %s, %s, %s, %s)
        ''', (data['id_local'], data['provincia'], data['ciudad'],
data['calle'], data['nombre']))
        conn.commit()
        return jsonify({"message": "Local creado exitosamente"}), 201
    except Exception as e:
        return jsonify({"error": f"Error al crear el local: {str(e)}"}), 500
    finally:
        if 'cur' in locals():
            cur.close()
        if 'conn' in locals():
            conn.close()
```



POST ▼ http://127.0.0.1:5000/local/

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

```
1 {  
2     "calle": "Passeig de Gràcia",  
3     "ciudad": "Barcelona",  
4     "id_local": 2,  
5     "nombre": "Store B",  
6     "provincia": "Barcelona"  
7 }
```

Status: 201 CREATED Size: 40 Bytes Time: 296 ms

```
1 {  
2     "message": "Local creado exitosamente"  
3 }
```

PUT

```
Python  
@local_routes.route('/<int:id_local>', methods=['PUT'])  
def update_local(id_local):  
    data = request.json  
    try:  
        conn = get_db_connection()  
        cur = conn.cursor()  
  
        # Recuperar los datos actuales del local  
        cur.execute('SELECT provincia, ciudad, calle, nombre FROM LOCAL WHERE  
id_local = %s;', (id_local,))  
        local = cur.fetchone()  
        if not local:  
            return jsonify({"error": "Local no encontrado"}), 404  
  
        # Actualizar solo los campos enviados en el cuerpo  
        provincia = data.get('provincia', local[0])
```



```
ciudad = data.get('ciudad', local[1])
calle = data.get('calle', local[2])
nombre = data.get('nombre', local[3])

cur.execute("""
    UPDATE LOCAL
    SET provincia = %s, ciudad = %s, calle = %s, nombre = %s
    WHERE id_local = %s
    ''', (provincia, ciudad, calle, nombre, id_local))
conn.commit()
return jsonify({"message": "Local actualizado exitosamente"}), 200
except Exception as e:
    return jsonify({"error": f"Error al actualizar el local: {str(e)}"}), 500
finally:
    if 'cur' in locals():
        cur.close()
    if 'conn' in locals():
        conn.close()
```

PUT ▾ http://127.0.0.1:5000/local/2

Query Headers² Auth Body¹ Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

```
1  {
2      "nombre": "Store B Updated"
3  }
```

Status: 200 OK Size: 45 Bytes Time: 283 ms

```
1  {
2      "message": "Local actualizado exitosamente"
3  }
```



DELETE

Python

```
@local_routes.route('/<int:id_local>', methods=['DELETE'])
def delete_local(id_local):
    try:
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute('DELETE FROM LOCAL WHERE id_local = %s;', (id_local,))
        conn.commit()
        if cur.rowcount == 0:
            return jsonify({"error": "Local no encontrado"}), 404
        return jsonify({"message": "Local eliminado exitosamente"}), 200
    except Exception as e:
        return jsonify({"error": f"Error al eliminar el local: {str(e)}"}), 500
    finally:
        if 'cur' in locals():
            cur.close()
        if 'conn' in locals():
            conn.close()
```

The screenshot shows the Postman interface with the following details:

- Method: DELETE
- URL: `http://127.0.0.1:5000/local/2`
- Query tab selected.
- Status: 200 OK
- Size: 43 Bytes
- Time: 285 ms
- Response body:

```
1  {
2      "message": "Local eliminado exitosamente"
3 }
```

Estos son ejemplos de las funciones CRUD para la tabla “Local”, el resto son muy similares, lo que suele cambiar es la lógica en la consulta y poco más, por tanto estos ejemplos son representativos de como son las funciones dentro de la api.



Presupuesto

Concepto	Descripción	Coste en euros
Personal	Trabajo de 2 ingenieros durante 5 días (40 horas/persona) a €35/hora/persona.	2800
Software	Uso de herramientas gratuitas como PostgreSQL, Draw.io	0
Hardware	Servidor para pruebas (5 días a €10/día).	50
Materiales	Material auxiliar (almacenamiento, cables, etc.).	50
Total	Incluyendo todos los costes descritos.	2900

Conclusiones

El proyecto ha resultado en una base de datos robusta, escalable y eficiente, con operaciones CRUD completas y consultas adaptables a diversas necesidades. La herencia en productos mejora la flexibilidad del modelo, mientras que la API REST garantiza integración y automatización. Las validaciones y restricciones refuerzan la consistencia y seguridad de los datos.