Proyecto Integrador: Generador de inserciones SQL desde facturas de supermercado

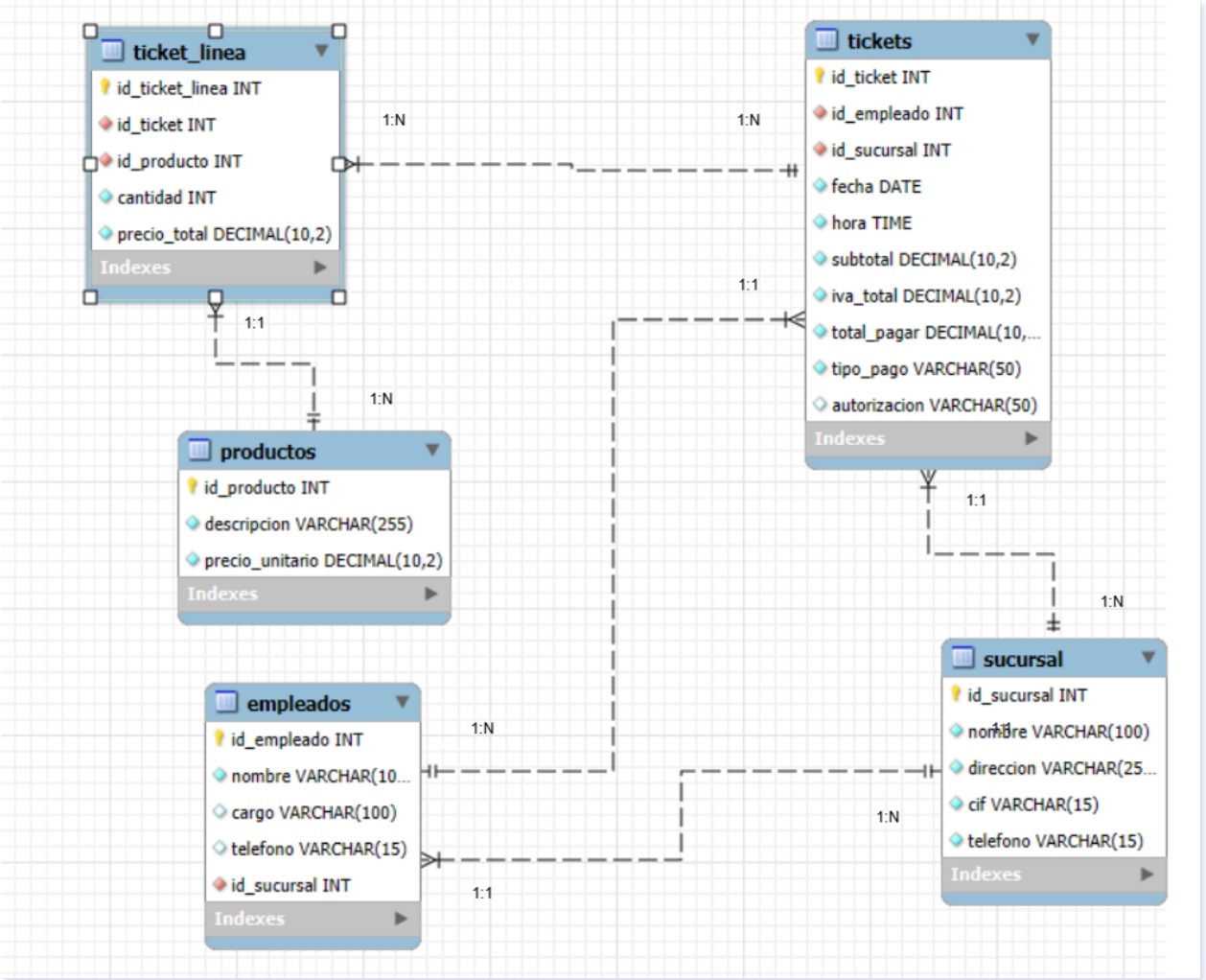
Por: María Chaparro Caballero

GIT: MChaparroCaballero

GRADO : Desarrollo de aplicaciones Web

ÍNDICE

1.Diagrama


ticket_linea
id_ticket_linea INT
id_ticket INT
id_producto INT
cantidad INT
precio_total DECIMAL(10,2)
Indexes
tickets
id_ticket INT
id_empleado INT
id_sucursal INT
fecha DATE
hora TIME
subtotal DECIMAL(10,2)
iva_total DECIMAL(10,2)
total_pagar DECIMAL(10,...
tipo_pago VARCHAR(50)
autorizacion VARCHAR(50)
Indexes
productos
id_producto INT
descripcion VARCHAR(255)
precio_unitario DECIMAL(10,2)
Indexes
empleados
id_empleado INT
nombre VARCHAR(10...
cargo VARCHAR(100)
telefono VARCHAR(15)
id_sucursal INT
Indexes
sucursal
id_sucursal INT
nombre VARCHAR(100)
direccion VARCHAR(25...
cif VARCHAR(15)
telefono VARCHAR(15)
Indexes
1:N
1:N
1:1
1:N
1:1
1:1
1:N
1:N
1:N
1:1

1. Explicación de la base de datos:

1.1. Introducción de tablas:

La base de datos supermercado está compuesta por cinco tablas principales:

“tickets” que contiene la información de cada ticket, “ticket_linea” que contiene cada producto almacenado en un ticket, “productos” que contiene todos los productos de la tienda, “empleados” que contiene todos los empleados y “sucursal” que contiene todos los datos de cada tienda.

1.2 Tickets:

La tabla *Tickets* se relaciona con tres entidades fundamentales del negocio para registrar una venta completa. Se relaciona con *Empleados* (1:N) porque un único empleado puede generar múltiples tickets a lo largo del día. También se relaciona con *Sucursal* (1:N) debido a que una única sucursal es responsable de emitir muchos tickets de venta. Finalmente, se relaciona con *Ticket_Linea* (1:N), siendo esta la relación más crucial: un único registro de Tickets (la cabecera) debe estar asociado a muchos registros de Ticket_Linea (el detalle).

Campo	Tipo de Dato	Restricciones	Propósito
id_ticket	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único del ticket de venta.
id_empleado	INT	NOT NULL, FOREIGN KEY	Referencia al empleado que realizó la venta.
id_sucursal	INT	NOT NULL, FOREIGN KEY	Referencia a la sucursal donde se realizó la venta.
fecha	DATE	NOT NULL	Día en que se generó el ticket.
hora	TIME	NOT NULL	Hora exacta de la transacción.
subtotal	DECIMAL(10, 2)	NOT NULL	Suma de los artículos antes de impuestos.
iva_total	DECIMAL(10, 2)	NOT NULL	Monto total del impuesto (IVA) aplicado a la venta.
total_pagar	DECIMAL(10, 2)	NOT NULL	Importe final de la venta (subtotal + iva_total).
tipo_pago	VARCHAR(50)	NOT NULL	Forma en que se pagó (Ej: Efectivo, Tarjeta).
autorizacion	VARCHAR(50)		Código de autorización de la tarjeta (si aplica).

1.3 Ticket_linea:

La tabla *Ticket_Linea* sirve como el registro de los artículos vendidos en cada ticket.Se relaciona con *Tickets* mediante una cardinalidad N:1, lo que significa que múltiples líneas de detalle se agrupan bajo un único ticket. Al mismo tiempo, se relaciona con *Productos* también mediante N:1, lo que permite que una entrada específica del catálogo de productos (por ejemplo, "Leche entera") pueda ser referenciada o vendida muchas veces a lo largo de muchos tickets diferentes.

Campo	Tipo de Dato	Restricciones	Propósito
id_ticket_linea	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único de cada línea de detalle dentro de un ticket.
id_ticket	INT	NOT NULL, FOREIGN KEY	Clave foránea que vincula esta línea al ticket de venta principal.
id_producto	INT	NOT NULL, FOREIGN KEY	Clave foránea que identifica el producto específico comprado en esta línea.
cantidad	INT	NOT NULL	Número de unidades de ese producto que se compraron.
precio_total	DECIMAL(10, 2)	NOT NULL	Precio total de esta línea (generalmente cantidad multiplicada por el precio unitario).

1.4 Productos:

La tabla Productos es una tabla que se vincula únicamente con la tabla *Ticket_Linea* mediante una cardinalidad de 1:N (Uno a Muchos) para que un único registro de producto, como "Leche Desnatada" (el lado "Uno"), pueda ser vendido y registrado en muchas líneas de tickets diferentes (el lado "Muchos") a lo largo del tiempo.

Campo	Tipo de Dato	Restricciones	Propósito
id_producto	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único de cada producto en el catálogo.
descripcion	VARCHAR(255)	NOT NULL, UNIQUE	Nombre completo del producto (ej: "Leche Entera 1L"). Debe ser único.
precio_unitario	DECIMAL(10, 2)	NOT NULL	Precio de venta base por una unidad del producto.

1.5 Empleados:

Se vincula a *Sucursal* con una cardinalidad N:1, donde muchos empleados se asignan a una única sucursal. Por otro lado, se relaciona con la tabla *Tickets* con una cardinalidad 1:N; esto permite que un único empleado pueda ser el autor de muchos tickets de venta diferentes.

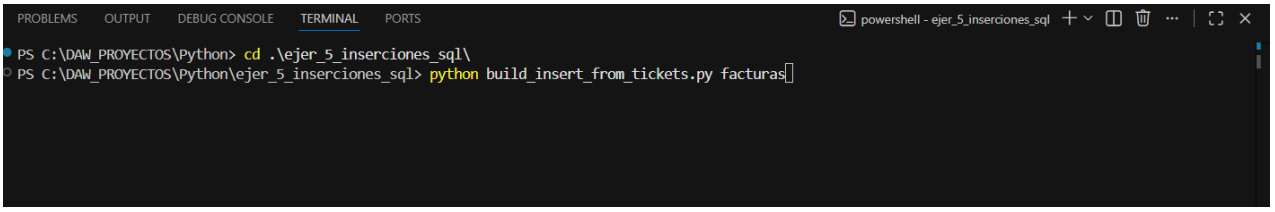
Campo	Tipo de Dato	Restricciones	Propósito
id_empleado	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único de cada empleado.
nombre	VARCHAR(100)	NOT NULL	Nombre completo del empleado.
cargo	VARCHAR(100)		Puesto o rol que desempeña el empleado (e.g., "Cajero", "Gerente").
telefono	VARCHAR(15)		Número de contacto del empleado.
id_sucursal	INT	NOT NULL, FOREIGN KEY	Clave foránea que indica la sucursal donde trabaja el empleado.

1.6 Sucursal:

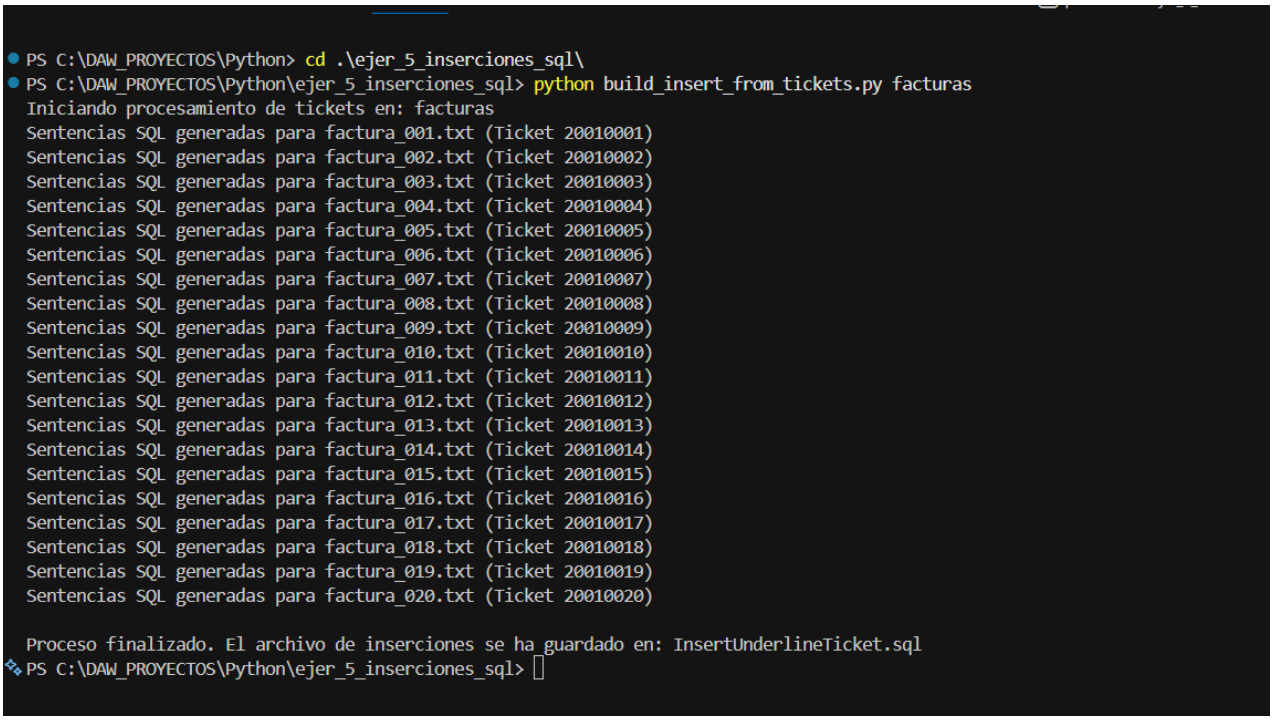
Primero, se relaciona con *Empleados* con una cardinalidad 1:N, lo que significa que una única sucursal es el lugar de trabajo de muchos empleados. Segundo, se relaciona con la tabla *Tickets* también mediante 1:N, lo que implica que una única sucursal puede ser el origen de muchos tickets de venta diferentes.

Campo	Tipo de Dato	Restricciones	Propósito
id_sucursal	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único de cada sucursal o tienda.
nombre	VARCHAR(100)	NOT NULL	Nombre comercial de la sucursal.
direccion	VARCHAR(255)	NOT NULL	Dirección física completa de la ubicación.
cif	VARCHAR(15)	UNIQUE, NOT NULL	Código de Identificación Fiscal de la sucursal (debe ser único).
telefono	VARCHAR(15)	NOT NULL	Número de teléfono de contacto de la sucursal.

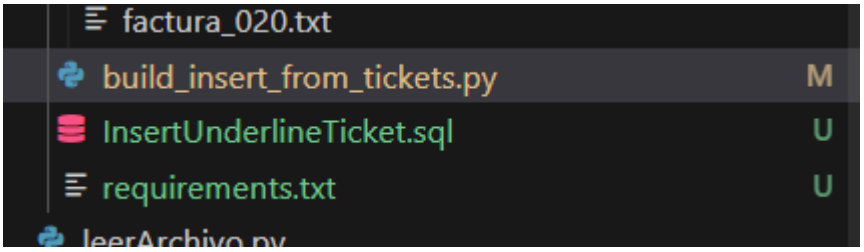
2. Comando



3. Salida



4. Archivo SQL creado.



er_5_inserciones_sql > InsertUnderlineTicket.sql

```sql

drop database if exists supermercado;
create database supermercado;
use supermercado;
DROP TABLE IF EXISTS Ticket_Linea;
DROP TABLE IF EXISTS Productos;
DROP TABLE IF EXISTS Tickets;
DROP TABLE IF EXISTS Empleados;
DROP TABLE IF EXISTS Sucursal;

CREATE TABLE Sucursal (
    id_sucursal INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    direccion VARCHAR(255) NOT NULL,
    cif VARCHAR(15) UNIQUE NOT NULL,
    telefono VARCHAR(15) NOT NULL
);

CREATE TABLE Empleados (
    id_empleado INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    cargo VARCHAR(100),
    telefono VARCHAR(15),
    id_sucursal INT NOT NULL,
    FOREIGN KEY (id_sucursal) REFERENCES Sucursal(id_sucursal)
);


CREATE TABLE Productos (
    id_producto INT AUTO_INCREMENT PRIMARY KEY,
    descripcion VARCHAR(255) NOT NULL UNIQUE,
    precio_unitario DECIMAL(10, 2) NOT NULL
);

```

```sql


CREATE TABLE Productos (
    id_producto INT AUTO_INCREMENT PRIMARY KEY,
    descripcion VARCHAR(255) NOT NULL UNIQUE,
    precio_unitario DECIMAL(10, 2) NOT NULL
);


CREATE TABLE Tickets (
    id_ticket INT PRIMARY KEY AUTO_INCREMENT,
    id_empleado INT NOT NULL,
    id_sucursal INT NOT NULL,
    fecha DATE NOT NULL,
    hora TIME NOT NULL,
    subtotal DECIMAL(10, 2) NOT NULL,
    iva_total DECIMAL(10, 2) NOT NULL,
    total_pagar DECIMAL(10, 2) NOT NULL,
    tipo_pago VARCHAR(50) NOT NULL,
    autorizacion VARCHAR(50),
    FOREIGN KEY (id_empleado) REFERENCES Empleados(id_empleado),
    FOREIGN KEY (id_sucursal) REFERENCES Sucursal(id_sucursal)
);

CREATE TABLE Ticket_Linea (
    id_ticket_linea INT AUTO_INCREMENT PRIMARY KEY,
    id_ticket INT NOT NULL,
    id_producto INT NOT NULL,
    cantidad INT NOT NULL,
    precio_total DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (id_ticket) REFERENCES Tickets(id_ticket),
    FOREIGN KEY (id_producto) REFERENCES Productos(id_producto)
);
```

```
);
START TRANSACTION;
INSERT IGNORE INTO Sucursal (nombre, direccion, cif, telefono) VALUES ('SUPERMERCADOS EL AHORRO', '123', 'B12345678', '91017

SELECT id_sucursal INTO @SucursalID
FROM Sucursal
WHERE cif = 'B12345678'
LIMIT 1;

INSERT IGNORE INTO Empleados (id_empleado, nombre, cargo, id_sucursal)
VALUES ('015', 'Juan Pérez', 'Cajero/a', @SucursalID);


SELECT id_empleado INTO @EmpleadoID
FROM Empleados
WHERE id_empleado = '015'
LIMIT 1;


INSERT INTO tickets (id_ticket, id_empleado, id_sucursal, fecha, hora, subtotal, iva_total, total_pagar, tipo_pago, autoriza
VALUES (
    20010001,
    @EmpleadoID,
    @SucursalID,
    '2025-09-11',
    '17:01',
    25.42,
    5.34,
    30.76,
    'TARJETA',
    '741486'
);
SET @TicketID = 20010001;
```

5. Explicación del código:

1. Verificamos que nos pasen la carpeta de facturas, después la almacenamos y luego creamos el archivo de salida.

```python
# Comprobamos que se ha pasado la carpeta de tickets
if len(sys.argv) != 2:
    print("Se requiere exactamente la carpeta de tickets")
    sys.exit(1)
#alamacenamos el nombre de la carpeta de tickets
carpeta_tickets = sys.argv[1]
# Archivo SQL de salida
archivo_sql_salida = 'InsertUnderlineTicket.sql'
```

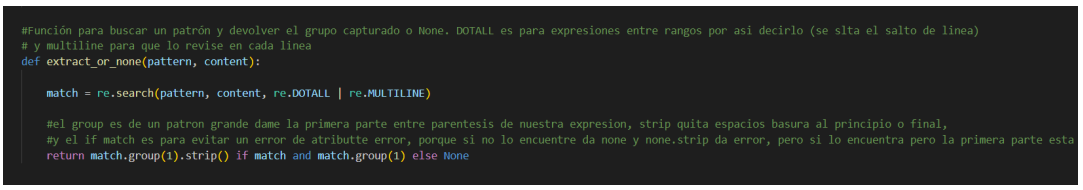
2. Creo la función generar_tablas()

Que me hace la parte del string que crea las tablas.

```python
#funcion que genera la parte del string que contiene la configuracio previa con sus drop si existen de la bd y tablas y las crea sino
def generar_tablas():
    sql_script = """
drop database if exists supermercado;
create database supermercado;
use supermercado;
DROP TABLE IF EXISTS Ticket_Linea;
DROP TABLE IF EXISTS Productos;
DROP TABLE IF EXISTS Tickets;
DROP TABLE IF EXISTS Empleados;
DROP TABLE IF EXISTS Sucursal;

CREATE TABLE Sucursal (
    id_sucursal INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    direccion VARCHAR(255) NOT NULL,
    cif VARCHAR(15) UNIQUE NOT NULL,
    telefono VARCHAR(15) NOT NULL
);

CREATE TABLE Empleados (
    id_empleado INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    cargo VARCHAR(100),
    telefono VARCHAR(15),
    id_sucursal INT NOT NULL,
    FOREIGN KEY (id_sucursal) REFERENCES Sucursal(id_sucursal)
);


CREATE TABLE Productos (
    id_producto INT AUTO_INCREMENT PRIMARY KEY,
    descripcion VARCHAR(255) NOT NULL UNIQUE,
    precio_unitario DECIMAL(10, 2) NOT NULL
);


CREATE TABLE Tickets (
    id_ticket INT PRIMARY KEY AUTO_INCREMENT,
    id_empleado INT NOT NULL,
    id_sucursal INT NOT NULL,
    fecha DATE NOT NULL,
    hora TIME NOT NULL,
    subtotal DECIMAL(10, 2) NOT NULL,
    iva_total DECIMAL(10, 2) NOT NULL,
    total_pagar DECIMAL(10, 2) NOT NULL,
    tipo_pago VARCHAR(50) NOT NULL,
    autorizacion VARCHAR(50),
    FOREIGN KEY (id_empleado) REFERENCES Empleados(id_empleado),
    FOREIGN KEY (id_sucursal) REFERENCES Sucursal(id_sucursal)
);

CREATE TABLE Ticket_Linea (
    id_ticket_linea INT AUTO_INCREMENT PRIMARY KEY,
    id_ticket INT NOT NULL,
    id_producto INT NOT NULL,
    cantidad INT NOT NULL,
    precio_total DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (id_ticket) REFERENCES Tickets(id_ticket),
    FOREIGN KEY (id_producto) REFERENCES Productos(id_producto)
);
"""
    return sql_script
```

3. Creo la función extract_or_none()

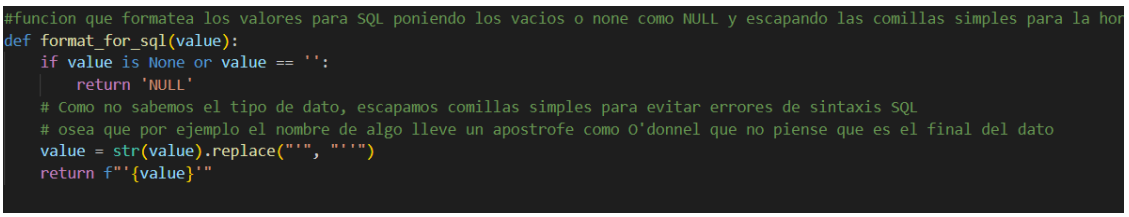
Para que, si no se encuentra un patrón usando el módulo re junto con DOTALL (para saltar saltos de línea) y MULTILINE (para que mire línea a línea), devuelva none, y si lo encuentra, nos quitará los espacios en blanco de delante y atrás con .strip() y devolverá el primer grupo del patrón grande buscado.



```python
#Función para buscar un patrón y devolver el grupo capturado o None. DOTALL es para expresiones entre rangos por asi decirlo (se slta el salto de linea)
# y multiline para que lo revise en cada linea
def extract_or_none(pattern, content):

    match = re.search(pattern, content, re.DOTALL | re.MULTILINE)

    #el group es de un patron grande dame la primera parte entre parentesis de nuestra expresion, strip quita espacios basura al principio o final,
    #y el if match es para evitar un error de atributte error, porque si no lo encuentre da none y none.strip da error, pero si lo encuentra pero la primera parte esta
    return match.group(1).strip() if match and match.group(1) else None
```

4. Creo la función format_for_sql()

Que formatea para quitar las comillas simples (apóstrofes) o cambiar el None a null en el string usando el replace.

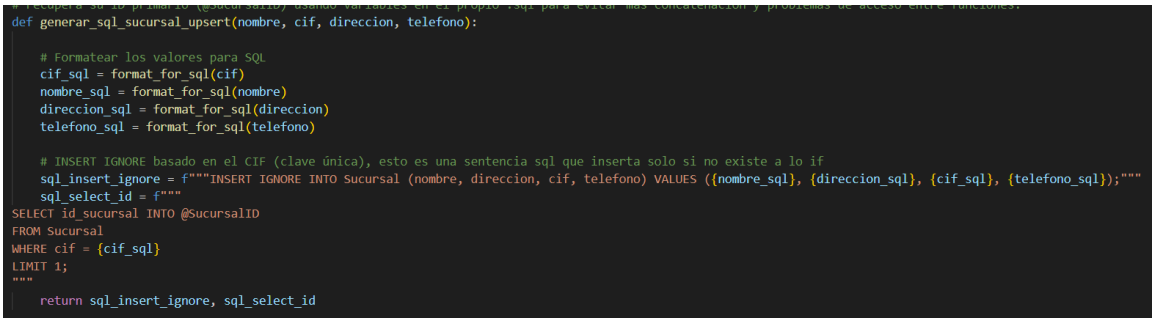


```python
#funcion que formatea los valores para SQL poniendo los vacios o none como NULL y escapando las comillas simples para la hor
def format_for_sql(value):
    if value is None or value == '':
        return 'NULL'
    # Como no sabemos el tipo de dato, escapamos comillas simples para evitar errores de sintaxis SQL
    # osea que por ejemplo el nombre de algo lleve un apostrofe como O'donnel que no piense que es el final del dato
    value = str(value).replace("'", "''")
    return f"'{value}'"
```

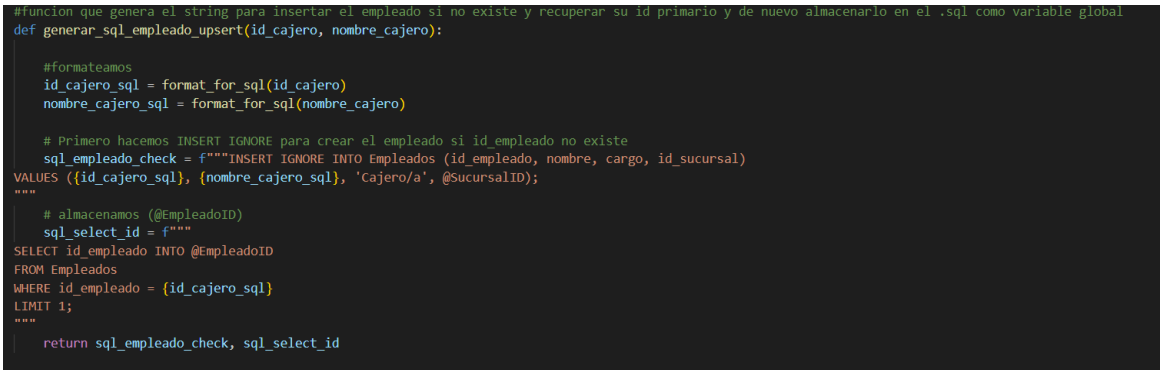
5. Creo un conjunto de funciones UPSERT

Para cada una de las tablas que actúan como upsert utilizando INSERT IGNORE para que, si no existe un registro con esos datos, me lo cree.

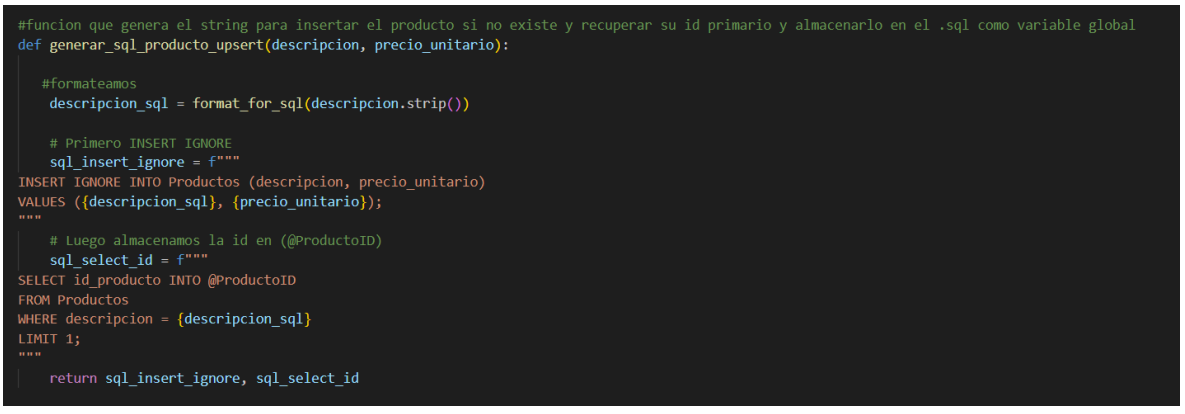
En otras palabras, estas funciones, cuando intentemos crear un ticket, nos crearán a la vez en las respectivas tablas los registros que no existieran. Por otro lado tendremos que formatear sus campos con *format_for_sql()* y además almacenaremos la id del registro en variables dentro del propio MySQL para que sean globales a la hora de crear el archivo a todas las tablas.



```
def generar_sql_sucursal_upsert(nombre, cif, direccion, telefono):

    # Formatear los valores para SQL
    cif_sql = format_for_sql(cif)
    nombre_sql = format_for_sql(nombre)
    direccion_sql = format_for_sql(direccion)
    telefono_sql = format_for_sql(telefono)

    # INSERT IGNORE basado en el CIF (clave única), esto es una sentencia sql que inserta solo si no existe a lo if
    sql_insert_ignore = f"""INSERT IGNORE INTO Sucursal (nombre, direccion, cif, telefono) VALUES ({nombre_sql}, {direccion_sql}, {cif_sql}, {telefono_sql});"""
    sql_select_id = f"""
SELECT id_sucursal INTO @SucursalID
FROM Sucursal
WHERE cif = {cif_sql}
LIMIT 1;
"""
    return sql_insert_ignore, sql_select_id
```



```
#funcion que genera el string para insertar el empleado si no existe y recuperar su id primario y de nuevo almacenarlo en el .sql como variable global
def generar_sql_empleado_upsert(id_cajero, nombre_cajero):

    #formateamos
    id_cajero_sql = format_for_sql(id_cajero)
    nombre_cajero_sql = format_for_sql(nombre_cajero)

    # Primero hacemos INSERT IGNORE para crear el empleado si id_empleado no existe
    sql_empleado_check = f"""INSERT IGNORE INTO Empleados (id_empleado, nombre, cargo, id_sucursal)
VALUES ({id_cajero_sql}, {nombre_cajero_sql}, 'Cajero/a', @SucursalID);
"""
    # almacenamos (@EmpleadoID)
    sql_select_id = f"""
SELECT id_empleado INTO @EmpleadoID
FROM Empleados
WHERE id_empleado = {id_cajero_sql}
LIMIT 1;
"""
    return sql_empleado_check, sql_select_id
```



```
#funcion que genera el string para insertar el producto si no existe y recuperar su id primario y almacenarlo en el .sql como variable global
def generar_sql_producto_upsert(descripcion, precio_unitario):

   #formateamos
    descripcion_sql = format_for_sql(descripcion.strip())

    # Primero INSERT IGNORE
    sql_insert_ignore = f"""
INSERT IGNORE INTO Productos (descripcion, precio_unitario)
VALUES ({descripcion_sql}, {precio_unitario});
"""
    # Luego almacenamos la id en (@ProductoID)
    sql_select_id = f"""
SELECT id_producto INTO @ProductoID
FROM Productos
WHERE descripcion = {descripcion_sql}
LIMIT 1;
"""
    return sql_insert_ignore, sql_select_id
```

6. Creo la función ***parsear_ticket***

Para que con expresiones regulares me encuentre en el ticket cada campo (llamando aquí la función ***extract_or_none*** para controlar la falta de un valor de un campo, como podría ser de autorización, que solo existe si el pago ha sido con tarjeta, para que lo ponga como null y no colapse la inserción) y devuelve un diccionario con los valores de los campos. Además, usaremos re.findall para que a la hora de buscar los productos en un ticket me pueda buscar varios con el mismo patrón.

```python
#funcion de parseo del ticket , es la que tiene todos los patrones regex para extraer los datos y
# devolver un diccionario con los datos usando a la de extract_or_none para evitar errores y si no hay dato que ponga None
def parsear_ticket(contenido_ticket):
    # Patrones para la Sucursal
    patron_nombre_sucursal = r"^\s*(.*?)\s*\n"
    patron_cif = r"CIF: (\w+)"
    patron_direccion = r"Av. Principal #(.*)- Madrid\s*\n"
    patron_telefono = r"Tel: (\d+)"

    # Patrones para el Ticket y Detalle
    patron_numero_ticket = r"Ticket:\s(\d+)"
    patron_fecha = r"Fecha:\s*([^ ]+)"
    patron_hora = r"Hora:\s*([\d:]+)"
    patron_nombre_cajero = r"Cajero:\s*[\d\s-]*\s*([^\n]+)"
    patron_id_cajero = r"Cajero:\s*(\d+)"
    patron_subtotal = r"SUBTOTAL\s*(\d+\.\d{2})"
    patron_iva = r"IVA\s\(\d+%\)\s*(\d+\.\d{2})"
    patron_total_a_pagar = r"TOTAL A PAGAR\s*(\d+\.\d{2})"
    patron_forma_pago = r"FORMA DE PAGO:\s(\w+)"
    patron_autorizacion = r"Autorización:\s(\d+)"

    #  Aísla el bloque de productos, capturando todo entre las dos líneas de guiones
    patron_bloque_productos = r'CANT\s+DESCRIPCIÓN\s+IMPORTE\n-+\n(.*?)\n-+'

    # el patron
    patron_productos_linea = r'^\s*([\d\.,]+)\s+(.+?)\s+([\d]+\.[\d]{2})\s*$'


    try:
        #primera parte de diccionario de datos
        data = {
            'nombre_sucursal': extract_or_none(patron_nombre_sucursal, contenido_ticket),
            'cif_sucursal': extract_or_none(patron_cif, contenido_ticket),
            'direccion_sucursal': extract_or_none(patron_direccion, contenido_ticket),
            'telefono_sucursal': extract_or_none(patron_telefono, contenido_ticket),

            'id_ticket': extract_or_none(patron_numero_ticket, contenido_ticket),
            'fecha': extract_or_none(patron_fecha, contenido_ticket),
            'hora': extract_or_none(patron_hora, contenido_ticket),
            'cajero': extract_or_none(patron_nombre_cajero, contenido_ticket),
            'id_cajero': extract_or_none(patron_id_cajero, contenido_ticket),
            'subtotal': extract_or_none(patron_subtotal, contenido_ticket),
            'iva_total': extract_or_none(patron_iva, contenido_ticket),
            'total_pagar': extract_or_none(patron_total_a_pagar, contenido_ticket),
            'pago': extract_or_none(patron_forma_pago, contenido_ticket),
            'autorizacion_tarjeta': extract_or_none(patron_autorizacion, contenido_ticket),
        }


        # Aísla el texto que contiene solo el listado de productos con cantidades, descripciones e importes osea una linea del producto
        productos_texto = extract_or_none(patron_bloque_productos, contenido_ticket)
        #donde alamceanmos todas las lineas de productos del ticket
        productos_capturados = []

        if productos_texto:
            # re.findall para encontrar varios con el mismo patron, aqui encontramos todas las lineas de productos
            productos_capturados = re.findall(patron_productos_linea, productos_texto, re.MULTILINE)

        #lo agregamos al diccionario
        data['productos'] = productos_capturados

        # Validación de campos críticos, si esto esta mal nos da none
        if not all([data['id_ticket'], data['fecha'], data['total_pagar']]):
            print(f"Error: El ticket {data.get('id_ticket', 'sin número')} no contiene datos esenciales.")
            return None

        return data

    except Exception as e:
        print(f"Error al parsear el ticket: {e}")
```

7. Con ***generar_sql_insert***

Almaceno en un array todos los strings de inserción y creación a los que simplemente iré añadiendo. Primero preparo la fecha en el formato que utiliza MySQL de YYYY-mm-dd; para ello, primero convierto la fecha del diccionario a un date con strptime del módulo datetime y después convierto ese datetime de nuevo a string, ahora ya con el formato correcto a Y-m-d con strftime.

```python
#funcion que genera las inserciones sql para un ticket dado los datos parseados usando upserts y transacciones
def generar_sql_insert(data, filename):

    #lista donde almacenaremos las sentencias sql
    sql_statements = []

    #esto como va a funcionar es que vamos a ir generando las sentencias sql y las vamos añadiendo a la lista
    #luego al final las unimos todas con saltos de linea y las devolvemos


    # Preparar la Fecha que como viene en dd/mm/yyyy hay que convertirla a yyyy-mm-dd y para eso usamos datetime
    fecha_mysql = 'NULL'
    if data['fecha']:
        try:
            #parsear el string de la fecha del formato dd/mm/yyyy al formato yyyy-mm-dd que es el de mysql
            fecha_obj = datetime.datetime.strptime(data['fecha'], '%d/%m/%Y')#usamos strptime para poner el formato que queremos y hacerlo datetime


            #ahora del objeto convertimos en texto EN EL FORMATO QUE QUEREMOS PARA SQL
            fecha_mysql = f"'{fecha_obj.strftime('%Y-%m-%d')}'"#usamos strftime para pasarlo a texto
        except (ValueError, TypeError):

            #si hay error en el formato de la fecha lanzamos una excepcion
            raise ValueError(f"Error de formato: La fecha '{data.get('fecha')}' del ticket no coincide con el formato esperado (DD/MM/YYYY).")

    # iniciamos la transaccion aka o todas se jecutan o ninguna osea o todas las inserciones A LAS DISTINTAS TABLAS se hacen o ninguna
    sql_statements.append("START TRANSACTION;")
```

Después añado al string que iniciamos una transacción en el .sql para que a la hora de crearme los respectivos campos en cada tabla, o sea exitoso en todos o falle completamente.

Tras eso, continúo con los distintos upserts(***generar_sql_empleado_upsert*** y ***generar_sql_sucursal_upsert***) en los que les pongo valores por defecto a los datos que sí eran not null en la tabla en caso de no tener valor y los añadimos al array.

```python
# hacemos los UPSERT de Sucursal
if data.get('cif_sucursal'):

    #recordemos primero el insert y luego el id
    sql_s_insert, sql_s_select = generar_sql_sucursal_upsert(
        data['nombre_sucursal'],
        data['cif_sucursal'],
        #usamos operador ternario para poner valores por defecto
        #si no hay datos que para poner o un dato vacio o otro nos lo podemos permitir y queda más llimpio
        data['direccion_sucursal'] if data.get('direccion_sucursal') else 'Direccion Desconocida',
        data['telefono_sucursal'] if data.get('telefono_sucursal') else '000000000'
    )
    #lo añadimos a la lista de sentencias sql
    sql_statements.append(sql_s_insert)
    sql_statements.append(sql_s_select)
else:
    # Fallback: Si no hay CIF, asumimos que siempre es la sucursal por defecto ID=1 y lo añadimos a la lista de sentencias sql
    sql_statements.append("-- Datos de sucursal incompletos, usando ID 1 por defecto")
    sql_statements.append("SET @SucursalID = 1;")

# UPSERT de Empleado (Usa @SucursalID) de nuevo lo mismo que antes pero con el empleado
if data.get('id_cajero') and data.get('cajero'):
    sql_e_check, sql_e_select = generar_sql_empleado_upsert(data['id_cajero'], data['cajero'])
    #lo añadimos a la lista de sentencias sql
    sql_statements.append(sql_e_check)
    sql_statements.append(sql_e_select)
else:
    # Fallback si no hay datos de cajero válidos
    sql_statements.append("-- Datos de cajero incompletos, usando ID 1 por defecto")
    sql_statements.append("SET @EmpleadoID = 1;")
```

Después hacemos la parte de string de los tickets usando nuestras variables de mysql globales y formateando los textos con control para null

```
        sql_statements.append("SET @EmpleadoID = 1;")

    # Iniciamos la inserción del Ticket, usando @EmpleadoID y @SucursalID y formateando los valores para SQL,
    #  de el diccionario previamente creado, y nos guardamos el id_ticket en @TicketID para usarlo en las lineas de detalle
    sql_ticket = f"""
INSERT INTO tickets (id_ticket, id_empleado, id_sucursal, fecha, hora, subtotal, iva_total, total_pagar, tipo_pago, autorizacion)
VALUES (
    {data['id_ticket']},
    @EmpleadoID,
    @SucursalID,
    {fecha_mysql},
    {format_for_sql(data['hora'])},
    {data['subtotal'] if data.get('subtotal') else 'NULL'},
    {data['iva_total'] if data.get('iva_total') else 'NULL'},
    {data['total_pagar']},
    {format_for_sql(data.get('pago'))},
    {format_for_sql(data.get('autorizacion_tarjeta'))}
);
SET @TicketID = {data['id_ticket']};
"""
    sql_statements.append(sql_ticket)#  la agregamos a la lista de sentencias sql
```

Preparamos nuestros datos de ticket_linea con round para los precios por cada producto según su cantidad en el ticket (precio_unitario), y limpiamos cantidad, convirtiéndola en int para que no haya decimales que no debería, manejando errores de división por 0, y creamos el producto si no existe con ***generar_sql_producto_upsert.***

```
# Líneas de Detalle del Ticket
for cantidad, descripcion, importe_total in data['productos']:

    try:
        # Eliminar todos los separadores de miles (punto y coma)
        cantidad_limpia_str = cantidad.replace('.', '').replace(',', '')

        # Convertir a entero (int) para reflejar que no hay decimales,
        #  importante para cantidades que si no el programa se piensa que hay miles
        cantidad = int(cantidad_limpia_str)

    except (ValueError):#si no se puede convertir a int
        # Si falla el parseo (es texto basura o vacío), la cantidad es 1
        cantidad = 1
        print(f"Advertencia: Cantidad inválida '{cantidad}'. Asignada a 1.")
    except (TypeError):#si es None
        cantidad = 1
        print(f"Advertencia: Cantidad inválida '{cantidad}'. Asignada a 1.")

    # Calcular precio unitario
    try:
        precio_unitario = round(float(importe_total) / cantidad, 2)
    except (ValueError):#si no se puede convertir a float
        precio_unitario = 0.00
    except (ZeroDivisionError):#si se intenta dividir por 0
        precio_unitario = 0.00

    #Generar sentencias de UPSERT de Producto (SET @ProductoID)
    sql_insert_p, sql_select_p_id = generar_sql_producto_upsert(descripcion.strip(), precio_unitario)

    #añadimos las sentencias a la lista
    sql_statements.append(sql_insert_p)
    sql_statements.append(sql_select_p_id)

    # hacemos la insercion de la linea del ticket usando las variables almacenadas @TicketID y @ProductoID
    sql_detalle = f"""
```

Por último, antes de empezar con el bucle principal de código, hacemos la sentencia del ticket y además agregamos un commit al array de string para que ejecute la transacción en el .sql y usamos join para añadir saltos de líneas entre las sentencias.

```
        sql_statements.append(sql_select_p_id)

        # hacemos la insercion de la linea del ticket usando las variables almacenadas @TicketID y @ProductoID
        sql_detalle = f"""
INSERT INTO ticket_Linea (id_ticket, id_producto, cantidad, precio_total)
VALUES (
    @TicketID,
    @ProductoID,
    {cantidad},
    {float(importe_total)}
);
"""
        #añadimos la sentencia a la lista
        sql_statements.append(sql_detalle)

    # Finalizar Transacción
    sql_statements.append("COMMIT;\n")

    return "\n".join(sql_statements)#añadimos saltos de lineas y las separamos
```

8. Por último, en la parte principal del código:

8.1. Informo de que empezamos el proceso.

```
#codigo principal
try:
    #primero informamos de donde vamos a procesar los tickets simplemente informativo para que sepamos si ha empezado
    #siendo algo con tanta escritura de archivos podria tardar un poco asi que informamos de que empieza
    print(f"Iniciando procesamiento de tickets en: {carpeta_tickets}")
```

8.2. Compruebo si ya existe el archivo de salida.

Para que, si existe, no me lo cree de nuevo.

```
#si no existe lo creamos
if not os.path.exists(archivo_sql_salida):
    # Abre el archivo SQL de salida en modo escritura ('w')
```

8.3. Abro el archivo donde pretendemos escribir.

En modo escritura y escribo las tablas en él.

```
    # Abre el archivo SQL de salida en modo escritura ('w')
    with open(archivo_sql_salida, 'w', encoding='utf-8') as sql_file:

        #genera las tablas y el use
        sql_file.write(generar_tablas())
```

8.4. Después me comprueba si el nombre del archivo está en la lista de las facturas con listdir y en orden con sorted.

```
#listdir devuelve una lista con los nombres de los archivos en la
# asi que procesamos los tickets en orden comprobando que esten d
for filename in sorted(os.listdir(carpeta_tickets)):

    #construye la ruta completa del archivo
    filepath = carpeta_tickets + "/" + filename
```

8.5. Luego, por cada ticket dentro de la carpeta de facturas

Va leyendo todo el contenido del ticket, parseando (extrayendo los campos) y creando los tickets y sus derivados para ir escribiéndolos en el archivo .sql, manejando errores específicos como ZeroDivision o ValueError y errores generales, y doy un mensaje de éxito si todas han sido completadas.

```
            #abrimos el ticket(va ticket por ticket)
            try:
                with open(filepath, 'r', encoding='utf-8') as f:
                    #lee todo el contenido del ticket y lo almacena en una variable
                    contenido_ticket = f.read()

                #parsea el ticket para extraer los datos
                datos_ticket = parsear_ticket(contenido_ticket)

                #si se han extraido datos correctamente, genera las inserciones SQL
                if datos_ticket:
                    #alacenar las inserciones SQL generadas en una variable
                    sql_inserts = generar_sql_insert(datos_ticket, filename)

                    #escribe las inserciones SQL en el archivo de salida
                    sql_file.write(sql_inserts)
                    sql_file.write("\n")

                    print(f"Sentencias SQL generadas para {filename} (Ticket {datos_ticket['id_ticket']})")


            except FileNotFoundError:
                # Captura error si el archivo de ticket no se encuentra
                print(f" Archivo {filename} no encontrado o inaccesible. Saltando.")

            except ValueError as e:
                # Captura errores específicos como los que lanzaste en la función de parseo (ej. formato de fecha)
                print(f"No se pudo procesar el ticket {filename}. Detalle: {e}")

            except (ZeroDivisionError):#si se intenta dividir por 0
                print(f"Error de división por cero en el ticket {filename}. Saltando línea de producto.")

            except Exception as e:
                # Captura cualquier otro error no manejado (el genérico)
                print(f"Erro al procesar {filename}. Detalle: {e}")

        print(f"\nProceso finalizado. El archivo de inserciones se ha guardado en: {archivo_sql_salida}")
    #si ya existe el archivo me dices que no se crea
    else:
        print(f"Archivo '{archivo_sql_salida}' ya existe. No se creó de nuevo.")
#manejo de errores de carpeta no encontrada y general al intentar abrir el archivo sql de salida
except FileNotFoundError:
    print(f"No se encontró la carpeta de tickets ({carpeta_tickets}) o no se pudo crear el archivo SQL de salida.")
```