Universidad Nacional Autónoma de México

Facultad de ingeniería

Base de Datos

2022-1

Proyecto Base de datos

Grupo 01

Villarreal Maldonado José Fernando
Rosales Mendoza José Francisco
Silva Venegas Erick Saúl
González Alamilla Abraham Alexis

Introducción

La problemática que se nos presenta es diseñar una base de datos para una cadena de papelerías, donde se desea tener ciertos datos de la entidad proveedor, cliente, email, tienda, venta, producto. Se debe de tener un registro de los productos que se venden, precio al que se vendió el producto, artículos faltantes en la bodega y un registro de todas las ventas y regalos. Para la base de datos se usará postgres como manejador, se utilizara comandos PL/pgSQL para la solución de requerimientos del cliente. Se creará una interfaz gráfica amigable y sencilla para el usuario que interaccionará con la base de datos.

Para la conexión a postgres se exploró usar python, java. Nos decidimos por python por la facilidad que encontramos de entender cómo lograr la conexión entre el manejador y el lenguaje de programación, también fue el mejor documentado, por lo tanto el más sencillo de entender. Para la realización de la interfaz gráfica utilizamos HTML y flask ya que un integrante del equipo ya había trabajado antes en un proyecto similar. Utilizaremos las técnicas aprendidas en el curso para el diseño de la base de datos, así como, descomponer el enunciado de la problemática en verbos, relaciones y entidades. Tomar en cuenta los requerimientos que se nos dan. Considerar usar Modelo ER extendido y técnicas de recurrencia. Obtener el modelo Relacional siguiendo las reglas de conversión de un modelo ER a relacional.La creación de los comandos DDL que serán ingresados en nuestro manejador. La conexión más eficiente entre postgres y python para ingresar información por medio de un sitio web.

Plan de trabajo

- Generar el modelo entidad-relacional: Generar el modelo entidad-relacional para la visualización de las entidades y relaciones necesarias para solucionar el problema.
- Generar el modelo relacional para encontrar los diferentes tipos de datos que se utilizan en las entidades, así como para identificar cuales atributos deben implementar algún tipo de constraint. De igual manera se busca identificar en que relaciones es necesario crear una tabla intermedia y en cuales se deben crear llaves foráneas.
- Crear el código SQL necesario para la implementación de la base de datos dentro del RDBMS Postgres, lo que incluye creación de tablas, llaves principales y llaves foráneas.
- Conexión entre python y postgres.
- Implementación de la interfaz gráfica de la base de datos mediante HTML.
- Documentación de las tareas que se realizaron a lo largo del desarrollo de la base de datos.
- Exposición de los resultados obtenidos a lo largo del desarrollo del proyecto.

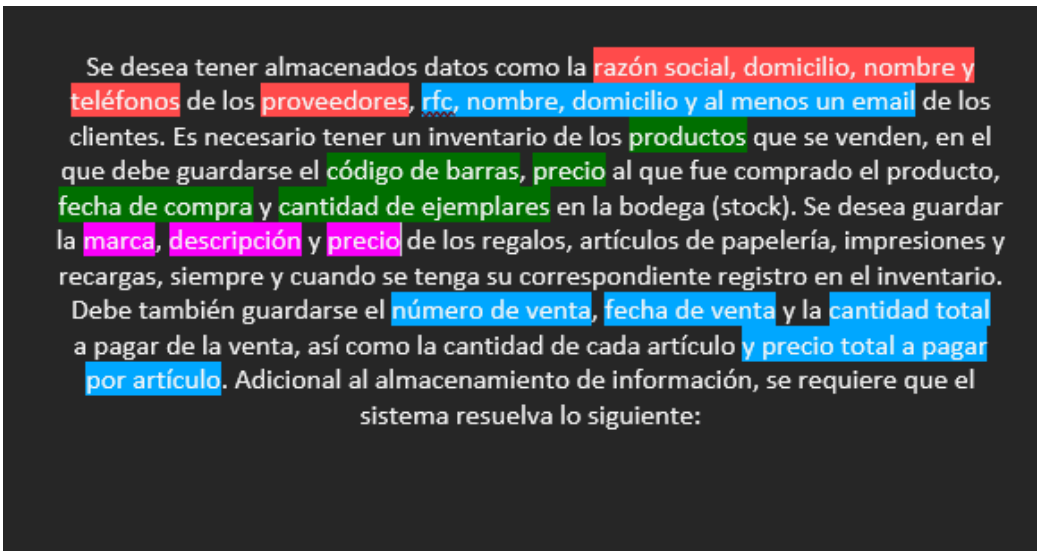
Desglose de actividades

Actividad	Alumno
1.Generar modelo entidad - relacion	Rosales Mendoza José Francisco
2.Generar modelo relacional	
2.1. Primera versión MR	Silva Venegas Erick Saúl
2.2. Version final	Rosales Mendoza José Francisco
3. Geneación DDL	
3.1. Primera versión	Silva Venegas Erick Saúl
3.2. Version final	Rosales Mendoza José Francisco
4. Creacion Datos.sql	Rosales Mendoza José Francisco
5.Programación PL/pgSQL (triggers y funciones)	Rosales Mendoza José Francisco
6. Conexión python y postgres	Rosales Mendoza José Francisco
7.Conexion html y python	Rosales Mendoza José Francisco
8..Creación de html	González Alamilla Abraham Alexis
	Rosales Mendoza José Francisco
9.Documentación	Rosales Mendoza José Francisco
	Villarreal Maldonado José Fernando
10.Presentacion	Rosales Mendoza José Francisco
	Silva Venegas Erick Saúl
11.Exposicion	Rosales Mendoza José Francisco

Diseño

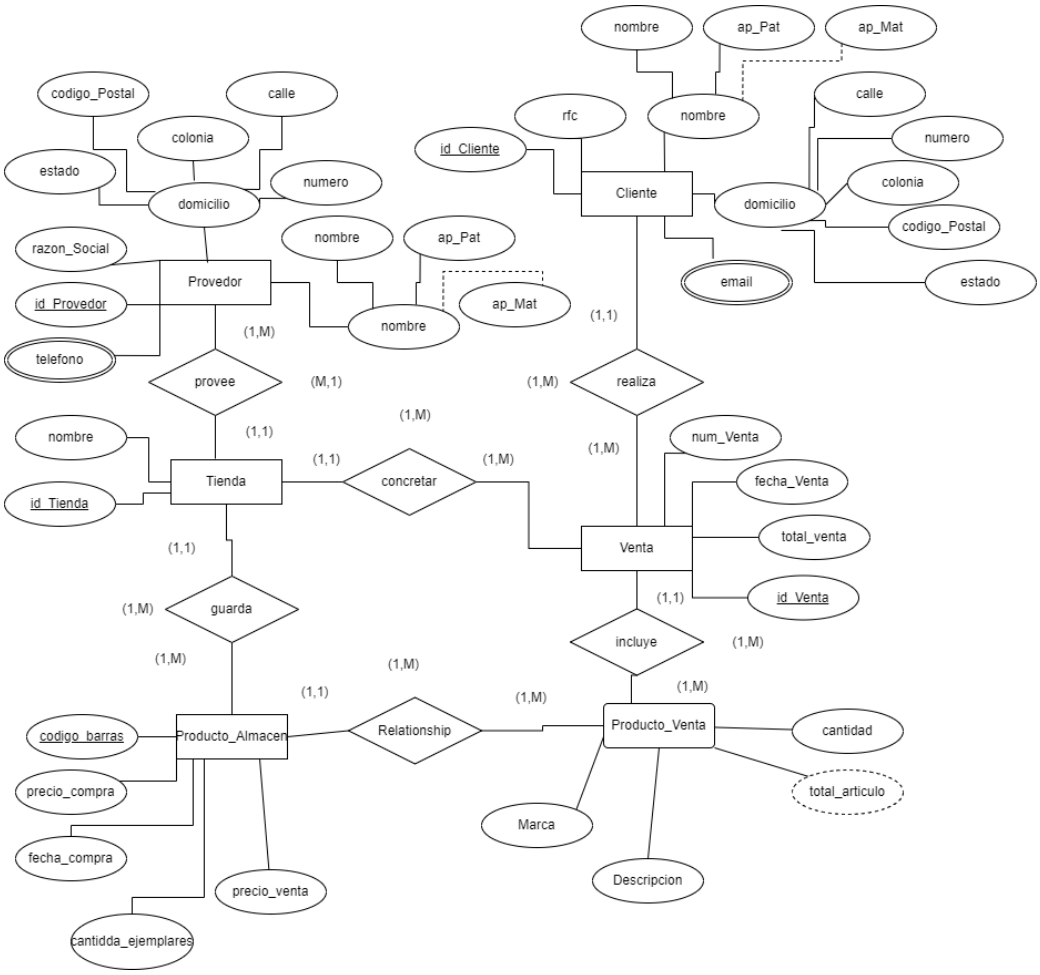
MODELO ENTIDAD RELACIÓN

Como primer etapa del diseño se realizo el modelo entidad relación, en el cual se identificaron 5 entidades principales con sus respectivos atributos:
-Proveedor
-Cliente
-Producto almacen
-Producto venta
-Venta



Y se considero que era necesario agregar un entidad tienda para conectar a todas las entidades anteriormente mencionadas.
-Tienda : id tienda, nombre

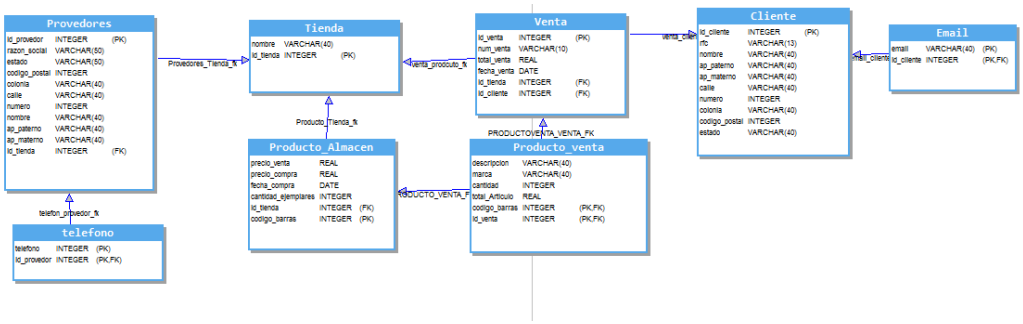
A continuación se muestra el diagrama MER



MODELO RELACIONAL

Como segunda etapa se desarrollo el modelo relacional partiendo del modelo entidad relación y por medio de ciertas reglas de conversión se obtuvo el siguiente modelo.

En el cual podemos observar que para los atributos multivaluados se creo una nueva tabla los cuales fueron teléfono y email, en los atributos opcionales no se puso la restricción de not null, en cada caso se definió con un cierto tipo de dato a consideración de lo más adecuado siendo la mayoría de tipo integer, varchar y date. Finalmente y como parte importante se propagaron las llaves foráneas en este caso para las cardinalidades 1:M se propago de la entidad con cardinalidad 1 a las M.



CREACIÓN BASE DE DATOS

Para la creación de la base de datos y con ayuda del software ER1 a partir de modelo relacional se autogenero el código DDL para la creación de tablas y sus respectivas llaves foraneas.

Finalmente en la implementación se crea la base de datos y se ejecuto el DDL para que se creen los objetos.

D: > Proyecto_BD > DDL.sql

```
4
5
6   CREATE DATABASE proyecto_bd;
7   \c proyecto_bd
8
9
10
11
12  CREATE TABLE PROVEDORES (
13      ID_PROVEDOR         INTEGER NOT NULL,
14      RAZON_SOCIAL        VARCHAR(50) NOT NULL,
15      ESTADO              VARCHAR(50) NOT NULL,
16      CODIGO_POSTAL       INTEGER NOT NULL,
17      COLONIA             VARCHAR(40) NOT NULL,
18      CALLE               VARCHAR(40) NOT NULL,
19      NUMERO              INTEGER NOT NULL,
20      NOMBRE              VARCHAR(40) NOT NULL,
21      AP_PATERNO          VARCHAR(40) NOT NULL,
22      AP_MATERNO          VARCHAR(40),
23      ID_TIENDA           INTEGER,
24  CONSTRAINT PK_PROVEDORES PRIMARY KEY (ID_PROVEDOR)
25  );
26
27  CREATE TABLE TIENDA (
28      NOMBRE              VARCHAR(40) NOT NULL,
29      ID_TIENDA           INTEGER NOT NULL,
30  CONSTRAINT PK_TIENDA PRIMARY KEY (ID_TIENDA)
31  );
32
33  CREATE TABLE PRODUCTO_ALMACEN (
34      PRECIO_VENTA        REAL NOT NULL,
```

```
postgres=# \i 'D:/Proyecto_BD/DDL.sql'
CREATE DATABASE
Ahora está conectado a la base de datos «proyecto_bd» con el usuario «postgres».
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
CREATE INDEX
proyecto_bd=#
```

-

```
proyecto_bd=# \d
             Listado de relaciones
 Esquema |      Nombre       | Tipo  |  Dueño
---------+-------------------+-------+----------
 public  | cliente           | tabla | postgres
 public  | email             | tabla | postgres
 public  | producto_almacen  | tabla | postgres
 public  | producto_venta    | tabla | postgres
 public  | provedores        | tabla | postgres
 public  | telefono          | tabla | postgres
 public  | tienda            | tabla | postgres
 public  | venta             | tabla | postgres
(8 filas)
```

Implementación

-DDL

Inicialmente se crea la base de datos y conecta a dicha base de datos

```
4
5
6    CREATE DATABASE proyecto_bd;
7    \c proyecto_bd
8
9
10
```

También ese DDL contiene las tablas y la restricción de llaves foráneas creadas automáticamente por el ER1.

```
12   CREATE TABLE PROVEDORES (
13     ID_PROVEDOR          INTEGER NOT NULL,
14     RAZON_SOCIAL         VARCHAR(50) NOT NULL,
15     ESTADO               VARCHAR(50) NOT NULL,
16     CODIGO_POSTAL        INTEGER NOT NULL,
17     COLONIA              VARCHAR(40) NOT NULL,
18     CALLE                VARCHAR(40) NOT NULL,
19     NUMERO               INTEGER NOT NULL,
20     NOMBRE               VARCHAR(40) NOT NULL,
21     AP_PATERNO           VARCHAR(40) NOT NULL,
22     AP_MATERNO           VARCHAR(40),
23     ID_TIENDA            INTEGER,
24   CONSTRAINT PK_PROVEDORES PRIMARY KEY (ID_PROVEDOR)
25   );
26
27   CREATE TABLE TIENDA (
28     NOMBRE               VARCHAR(40) NOT NULL,
29     ID_TIENDA            INTEGER NOT NULL,
30   CONSTRAINT PK_TIENDA PRIMARY KEY (ID_TIENDA)
31   );
32
33   CREATE TABLE PRODUCTO_ALMACEN (
34     PRECIO_VENTA         REAL NOT NULL,
35     PRECIO_COMPRA        REAL NOT NULL,
36     FECHA_COMPRA         DATE NOT NULL,
37     CANTIDAD_EJEMPLARES INTEGER NOT NULL,
38     ID_TIENDA            INTEGER,
39     CODIGO_BARRAS        INTEGER NOT NULL,
40   CONSTRAINT PK_PRODUCTO_ALMACEN PRIMARY KEY (CODIGO_BARRAS)
41   );
42
```

```sql
CREATE TABLE VENTA (
  ID_VENTA              INTEGER NOT NULL,
  NUM_VENTA             VARCHAR(10) NOT NULL,
  TOTAL_VENTA           REAL NOT NULL,
  FECHA_VENTA           DATE NOT NULL,
  ID_TIENDA             INTEGER,
  ID_CLIENTE            INTEGER,
CONSTRAINT PK_VENTA PRIMARY KEY (ID_VENTA)
);

CREATE TABLE CLIENTE (
  ID_CLIENTE            INTEGER NOT NULL,
  RFC                   VARCHAR(13) NOT NULL,
  NOMBRE                VARCHAR(40) NOT NULL,
  AP_PATERNO            VARCHAR(40) NOT NULL,
  AP_MATERNO            VARCHAR(40) NOT NULL,
  CALLE                 VARCHAR(40) NOT NULL,
  NUMERO                INTEGER NOT NULL,
  COLONIA               VARCHAR(40) NOT NULL,
  CODIGO_POSTAL         INTEGER NOT NULL,
  ESTADO                VARCHAR(40) NOT NULL,
CONSTRAINT PK_CLIENTE PRIMARY KEY (ID_CLIENTE)
);
```

```sql
CREATE TABLE PRODUCTO_VENTA (
  DESCRIPCION           VARCHAR(40) NOT NULL,
  MARCA                 VARCHAR(40) NOT NULL,
  CANTIDAD              INTEGER NOT NULL,
  TOTAL_ARTICULO        REAL NOT NULL,
  CODIGO_BARRAS         INTEGER NOT NULL,
  ID_VENTA              INTEGER NOT NULL,
CONSTRAINT PK_PRODUCTO_VENTA PRIMARY KEY (CODIGO_BARRAS,ID_VENTA)
);

CREATE TABLE TELEFONO (
  TELEFONO              INTEGER NOT NULL,
  ID_PROVEDOR           INTEGER NOT NULL,
CONSTRAINT PK_TELEFONO PRIMARY KEY (TELEFONO,ID_PROVEDOR)
);

CREATE TABLE EMAIL (
  EMAIL                 VARCHAR(40) NOT NULL,
  ID_CLIENTE            INTEGER NOT NULL,
CONSTRAINT PK_EMAIL PRIMARY KEY (EMAIL,ID_CLIENTE)
);
```

Se crean las restricciones de llaves foraneas:

```sql
/*==============================================================*/
/*                          FOREIGN KEYS                        */
/*==============================================================*/
ALTER TABLE PROVEDORES
    ADD CONSTRAINT FK_REFERENCE_1
        FOREIGN KEY (ID_TIENDA)
            REFERENCES TIENDA (ID_TIENDA) ON DELETE CASCADE ON UPDATE CASCADE
 ;


ALTER TABLE PRODUCTO_ALMACEN
    ADD CONSTRAINT FK_REFERENCE_2
        FOREIGN KEY (ID_TIENDA)
            REFERENCES TIENDA (ID_TIENDA) ON DELETE CASCADE ON UPDATE CASCADE
 ;


ALTER TABLE VENTA
    ADD CONSTRAINT FK_REFERENCE_3
        FOREIGN KEY (ID_TIENDA)
            REFERENCES TIENDA (ID_TIENDA) ON DELETE CASCADE ON UPDATE CASCADE
 ;

ALTER TABLE VENTA
    ADD CONSTRAINT FK_REFERENCE_4
        FOREIGN KEY (ID_CLIENTE)
            REFERENCES CLIENTE (ID_CLIENTE) ON DELETE RESTRICT ON UPDATE CASCADE
 ;
```

```sql
ALTER TABLE PRODUCTO_VENTA
    ADD CONSTRAINT FK_REFERENCE_8
        FOREIGN KEY (ID_VENTA)
            REFERENCES VENTA (ID_VENTA) ON DELETE RESTRICT ON UPDATE CASCADE
 ;

ALTER TABLE PRODUCTO_VENTA
    ADD CONSTRAINT FK_REFERENCE_9
        FOREIGN KEY (CODIGO_BARRAS)
            REFERENCES PRODUCTO_ALMACEN (CODIGO_BARRAS) ON DELETE RESTRICT ON UPDATE CASCADE
 ;


ALTER TABLE TELEFONO
    ADD CONSTRAINT FK_REFERENCE_7
        FOREIGN KEY (ID_PROVEDOR)
            REFERENCES PROVEDORES (ID_PROVEDOR) ON DELETE CASCADE ON UPDATE CASCADE
 ;


ALTER TABLE EMAIL
    ADD CONSTRAINT FK_REFERENCE_10
        FOREIGN KEY (ID_CLIENTE)
            REFERENCES CLIENTE (ID_CLIENTE) ON DELETE CASCADE ON UPDATE CASCADE
 ;
```

Se crea el índice que se pide en los requerimiento en id venta ya que un índice siempre va en la columna que es frecuentemente utilidad en búsquedas.

```sql
CREATE INDEX indice_producto_venta
ON producto_venta (id_venta);
```

ACCIONES.sql

Posteriormente al haber creado la base de datos con todas las restricciones necesarias se procede a ejecutar el archivo ACCIONES.sql que contiene funciones requeridas por el cliente con su referentes triggers si son necesarios,

además de algunos triggers extras a los requerimientos para llenar los atributos calculados. A continuación se explicara cada elemento del código:

regresa utilidad - Como funciona es recibiendo un id de un producto , después mediante una variable record se trae el registro que corresponde a ese id y de sus atributos precio compra y precio venta se calcula la utilidad en una variable de tipo real que es regresada y también mostrada mediante un mensaje en el manejador,

```
--- Regrese utilidad
CREATE FUNCTION regresa_utilidad(id_producto INTEGER ) RETURNS REAL AS $$
DECLARE producto record;
DECLARE utilidad REAL;
 BEGIN
    select * into producto from producto_almacen where codigo_barras = $1;
    RAISE NOTICE 'Utilidad = %',producto.precio_venta -producto.precio_compra;
    utilidad =  producto.precio_venta-producto.precio_compra;
    RETURN utilidad;
 END;
$$ LANGUAGE plpgsql;

```

veririfica Stock- Es una función que se activa cuando se inserta (antes de insertar) un nuevo producto venta, y lo que realiza es traer el registro de producto almacen con una variable record y hacer la diferencia entre cantidad de ejemplares de ese id y la cantidad de producto venta que se va vender, asignando el resultado a la variable cantidad actual y a patir de esta si es menor o igual que cero cancela la venta, si es mayor a 0 y menor a 3 realiza venta pero manda una alerta, y si no es ninguno de estos casos se realiza la venta.

```
CREATE FUNCTION veririfica_Stock() RETURNS trigger AS $$
DECLARE Producto_Stock record;
DECLARE cantidad_Actual INTEGER;
 BEGIN
    select * into Producto_Stock from producto_almacen where codigo_barras = NEW.codigo_barras;
    cantidad_Actual = Producto_Stock.cantidad_ejemplares - NEW.cantidad;
    IF cantidad_Actual = 0 AND  cantidad_Actual < 0THEN
        RAISE NOTICE 'No se completo la operación porque ya que nos quedamos sin productos';
        RETURN OLD;
    END IF;
    IF cantidad_Actual > 0 AND  cantidad_Actual < 3 THEN
        RAISE NOTICE 'Quedan menos de 3';
        UPDATE producto_almacen set cantidad_ejemplares = cantidad_Actual where codigo_barras = New.codigo_barras;
        RETURN NEW;
    END IF;
    UPDATE producto_almacen set cantidad_ejemplares = cantidad_Actual where codigo_barras = New.codigo_barras;
    RAISE NOTICE 'Transaccion completada numero actual % de producto %',cantidad_Actual,Producto_Stock.codigo_barras;
    RETURN NEW;
 END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER veririfica_Stock_trigger BEFORE INSERT ON producto_venta
FOR EACH ROW EXECUTE PROCEDURE veririfica_Stock();

```

veririfica menor 3 - Es un función que no recibe nada pero devuelve una tabla con las columnas de código de barras y cantidad, y lo que hace al entrar en el return es devolver un query de un select de la tabla producto almacen

con las columnas código barras y cantidad que cumplen con la condición de que la cantidad de sus ejemplares es menor a 3.

```
--- Funcion regresa de almacen menores a 3
CREATE FUNCTION veririfica_menor_3() RETURNS TABLE(codigo_barras integer, cantidad integer) AS $$
 BEGIN
    RETURN query Select p.codigo_barras,p.cantidad_ejemplares from producto_almacen p where p.cantidad_ejemplares<3 ;
 END;
$$ LANGUAGE plpgsql;
```

ventas rango código - Recibe dos datos de tipo date y regresa una tabla con la cantidad, total y el código de barras de los producto vendidos en cierto periodo, esto lo realiza regresando un query de un select donde se unen las tablas producto venta y venta a partir de id venta y se le aplica un filtrado con where y between para solo mostrar las estén en el rango entre las fechas pasada como parámetro, y el resulta lo agrupa según el código de barras y nos muestra las suma de la cantidad de la vendida y el total de ganancia de cada código de barras.

```
---FECHAS con codigo barras
CREATE FUNCTION ventas_rango_codigo(inicio date, fin date)
RETURNS TABLE(cantidad_f bigint, tota_f real, codigo_barras_f integer)
AS $$
BEGIN
    RETURN query select sum(cantidad) as cantidad_total, sum(total_articulo) as ganacia_total, codigo_barras from venta
    inner join producto_venta on venta.id_venta = producto_venta.id_venta
    where fecha_venta  between inicio and fin group by codigo_barras;
END;
$$ LANGUAGE plpgsql;
```

Esta función es casi idéntica pero aquí ya no se regresa el código de barras solo se regresa artículos vendidos y total de ganancias.

```
-- rango fechas cantidad y total
CREATE FUNCTION ventas_rango(inicio date, fin date)
RETURNS TABLE(cantidad_f bigint, tota_f real)
AS $$
BEGIN
    RETURN query select sum(cantidad) as cantidad_total, sum(total_articulo) as ganacia_total from venta
    inner join producto_venta on venta.id_venta = producto_venta.id_venta
    where fecha_venta  between inicio and fin;
END;
$$ LANGUAGE plpgsql;
```

generar vista - Una función que es activada por un trigger antes de insertar en la tabla producto venta la cual regresa un query que es un select de id venta,código barras, cantidad,total articulo,descripción,marca,fecha venta de la union de las tablas producto venta y ventan que cumplen con el filtra de que su id venta se igual al del producto que se esta ingresando.

```
---Genera factura
CREATE FUNCTION generar_vista() RETURNS trigger AS $$

 BEGIN
    CREATE OR REPLACE VIEW  factura
     AS
     SELECT venta.id_venta,codigo_barras, cantidad,total_articulo,descripcion,marca,fecha_venta FROM venta
     INNER join producto_venta on venta.id_venta = producto_venta.id_venta where venta.id_venta = 1;
     RETURN NEW;
 END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER generar_vista_trigger BEFORE INSERT ON venta
FOR EACH ROW EXECUTE PROCEDURE generar_vista();
```

calcula totales - Función que activa triggers antes de realizar inserción en producto venta y lo que realiza es traer un registro de producto almacen según el código de barras del producto que se va ingresar y con su atributo precio venta y cantidad que se esta ingresando se calcula el total del articulo. Y la segunda parte actualiza el total de venta a partir de traer su registro correspondiente de la tabla venta se le suma el total del articulo.

```sql
---EXTRAS

--- calcula el total del articulo
CREATE FUNCTION calcula_totales() RETURNS trigger AS $$
DECLARE producto record;
DECLARE aux record;
 BEGIN
    --Actualiza el total articulo
    SELECT * INTO producto FROM producto_almacen WHERE codigo_barras = NEW.codigo_barras;
    NEW.total_articulo = producto.precio_venta * NEW.cantidad;
    ---Actualiza el total venta
    SELECT * INTO aux FROM venta WHERE id_venta = NEW.id_venta;
    UPDATE venta set total_venta =  (aux.total_venta + (producto.precio_venta * NEW.cantidad)) WHERE id_venta = NEW.id_venta;

    RETURN NEW;
 END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER calcula_totales_trigger BEFORE INSERT ON producto_venta
FOR EACH ROW EXECUTE PROCEDURE calcula_totales();
```

-Datos.sql

Como ultimo se ejecuta Datos.sql que contiene algunos inserts de cada tipo de tabla para comenzar a hacer pruebas.

```sql
---Genera factura
CREATE FUNCTION generar_vista() RETURNS trigger AS $$

 BEGIN
    CREATE OR REPLACE VIEW  factura
     AS
     SELECT venta.id_venta,codigo_barras, cantidad,total_articulo,descripcion,marca,fecha_venta FROM venta
     INNER join producto_venta on venta.id_venta = producto_venta.id_venta where venta.id_venta = 1;
     RETURN NEW;
 END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER generar_vista_trigger BEFORE INSERT ON venta
FOR EACH ROW EXECUTE PROCEDURE generar_vista();
```


```sql
---EXTRAS

--- calcula el total del articulo
CREATE FUNCTION calcula_totales() RETURNS trigger AS $$
DECLARE producto record;
DECLARE aux record;
 BEGIN
    --Actualiza el total articulo
    SELECT * INTO producto FROM producto_almacen WHERE codigo_barras = NEW.codigo_barras;
    NEW.total_articulo = producto.precio_venta * NEW.cantidad;
    ---Actualiza el total venta
    SELECT * INTO aux FROM venta WHERE id_venta = NEW.id_venta;
    UPDATE venta set total_venta =  (aux.total_venta + (producto.precio_venta * NEW.cantidad)) WHERE id_venta = NEW.id_venta;

    RETURN NEW;
 END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER calcula_totales_trigger BEFORE INSERT ON producto_venta
FOR EACH ROW EXECUTE PROCEDURE calcula_totales();
```

Presentación

Para realiza la conexión se hizo uso un paquete psycopg2 que es especial para realizar la conexión con la base de datos..

La manera en que se estable la conexión es de la siguiente manera:

```
3   import psycopg2
4
5   app = Flask(__name__)
6
7
8
9   try:
10      Connection = psycopg2.connect(
11          host = 'localhost',
12          user = 'postgres',
13          password = '1234',
14          database = 'proyecto_bd'
15      )
16      print("conexion exitosa")
17      cursor = Connection.cursor()
18
19  except Exception as ex:
20      print(ex)
21
```

Primero se importa la librería psycopg2, después dentro de un try catch y mediante una función especial de psycopg2 se crea la conexión mandando como parámetros la información de la base de datos como lo es el host,usuario,la contraseña del usuario,y el nombre de la base datos y únicamente con eso se estable la conexión desde python con la base datos en este caso ´proyecto bd´.

Para realizar la acción de interacción entre la base de datos y python se ocupan

las siguientes lineas:

```
cursor = Connection.cursor()
```

```
VALUES (%s, %s, %s, %s, %s, %s,%s, %s, %s, %s);"""
cursor.execute(insertSQLcliente,(id_cliente,rfc,nombre,ap_paterno,ap_materno,estado,colonia,codigo_pos
print("deberia imprimir")
Connection.commit()
```

Donde se crea el cursor como se ve en la primera imagen. Y en la segunda imagen mediante su método execute se mando el query que uno desee y finalmente se hace commit que es para que la librería registre el cambio como permanente.

Pagina web

Para realiza la implementación de la pagina web que nos servirá como formulario se utilizo la librería flask como a continuación se explica:

Primero se definen ciertas propiedades base para realizar la conexión con las siguientes lineas:

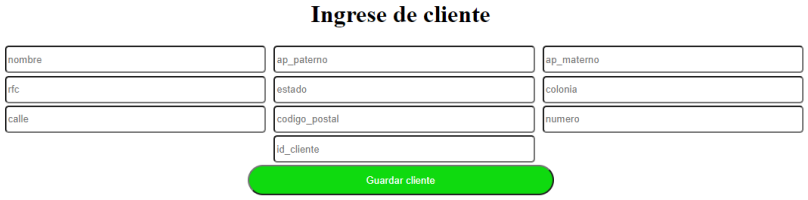
```
from flask import Flask, render_template, request
from flask.wrappers import Request
import psycopg2

app = Flask(__name__)
```

```
86
87    if __name__ == "__main__":
88        app.run(debug = True , port = 4000)
```

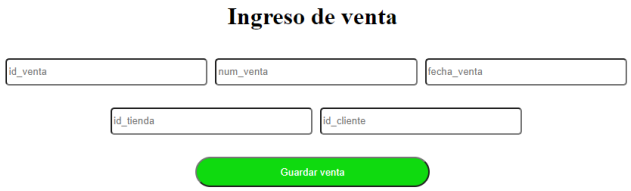
Una vez teniendo lo anterior se crean los bloques a partir de las posibles direcciones que puede tomar la pagina web, comenzando por la dirección raíz donde se renderizá el html cliente que nos permite obtener y llenar el formulario con los datos de el cliente:

```python
@app.route('/')
def Hola():
    return render_template('cliente.html')
```



A partir de esa función y ese html se da paso a la siguiente dirección Cliente en la cual se reciben los datos llenados en el formulario Cliente.html y se le da tratamiento insertandolos en la base de datos y renderizando el siguiente formulario para llenar los datos de la venta por medio del html venta.

```python
@app.route('/cliente', methods = ['POST'])
def cliente():
    if request.method == 'POST':
        nombre = request.form['nombre']
        ap_paterno = request.form['ap_paterno']
        ap_materno = request.form['ap_materno']
        rfc = request.form['rfc']
        estado = request.form['estado']
        colonia = request.form['colonia']
        codigo_postal = request.form['codigo_postal']
        calle = request.form['calle']
        numero = request.form['numero']
        id_cliente = request.form['id_cliente']
        insertSQLcliente = """INSERT INTO public.cliente(
        id_cliente, rfc, nombre, ap_paterno, ap_materno, estado, colonia, codigo_postal, calle, num
        VALUES (%s, %s, %s, %s, %s, %s,%s, %s, %s, %s);"""
        cursor.execute(insertSQLcliente,(id_cliente,rfc,nombre,ap_paterno,ap_materno,estado,colonia
        print("deberia imprimir")
        Connection.commit()
    return render_template('venta.html')
```

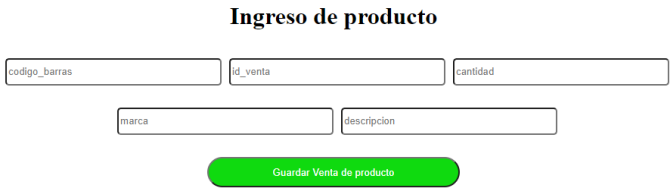


Después al registrar la venta se redirecciona a venta donde se reciben los datos de la venta, se insertan en la base datos y se renderiza el ultimo formulario para llenar los productos que contendrán esa venta.

```python
@app.route('/venta',methods = ['POST'])
def venta():
    if request.method == 'POST':
        id_tienda= request.form['id_tienda']
        fecha_venta = request.form['fecha_venta']
        num_venta = request.form['num_venta']
        id_venta = request.form['id_venta']
        id_cliente = request.form['id_cliente']

        insertSQLcliente = """INSERT INTO public.venta(
        id_venta, num_venta, total_venta, fecha_venta, id_tienda, id_cliente)
        VALUES
        (%s, %s, %s, %s, %s, %s);"""
        cursor.execute(insertSQLcliente,(id_venta,num_venta,0,fecha_venta,id_tienda,id_cliente))

        print("deberia imprimir")
        Connection.commit()
    return render_template('productoVenta.html')
```



De igual manera se llama a la función correspondiente y mediante cursor.execute se inserta la información recopilada por el html en la base datos.

```python
@app.route('/producto',methods = ['POST'])
def producto():
    if request.method == 'POST':
        codigo_barras= request.form['codigo_barras']
        id_venta = request.form['id_venta']
        cantidad = request.form['cantidad']
        marca = request.form['marca']
        descripcion = request.form['descripcion']

        insertSQLcliente = """INSERT INTO public.producto_venta(
        descripcion, marca, cantidad, total_articulo, codigo_barras, id_venta)
        VALUES
        (%s, %s, %s, %s , %s, %s);"""
        cursor.execute(insertSQLcliente,(descripcion,marca,cantidad,0,codigo_barras,id_venta))
        print("deberia imprimir")
        Connection.commit()
    return render_template('productoVenta.html')
```

Y se vuelve a renderizá el mismo html por si se desea agregar mas productos.

Lecciones aprendidas

Mediante la realización de este proyecto se logro comprender que es mas complejo de lo que parece el diseño de una base de datos desde el comienzo, porque una vez que ya teníamos ciertos modelos en etapas posteriores se lograba apreciar que el diseño no era muy adecuado por lo cual se tuvo que volver a realizar las primeras etapas de diseño con lo aprendido para dar mejor solución al problema que el cliente presento.

También se comprendió que es necesario tener una comunicación correcta con el cliente y entender lo mejor posible lo que el cliente quiere y además como ingenieros aprender a ser pro activos e indicarle al cliente posibles opciones que puedan resolver mejor el problema, así como también tomar decisiones en ciertos aspectos que se consideren pertinentes.

Otros aspectos que se comprendieron es que es muy útil realizar un buen diseño en cuanto a integridad referencial y manejo adecuado de la información ya que al momento en que el usuario haga uso es muy probable que cometa algún error para el cual la base de datos debe estar prepara.

Finalmente el mayor aprendizaje es que realizar el diseño de una base datos es menos complejo si se realiza por etapas, se utilizan herramientas de modelado y se siguen las reglas para manejar correctamente la información.