Proyecto Bases de Datos

Ana Laura Camacho Lara, Aldo Zetina Muciño,
Martínez Matías Joan Eduardo,a Vázquez Torres Juan Adrián.

August 2021

1 Introducción

Se nos solicitó implementar una solución para una cadena de papelerías esta solucion involucraba gestion de inventario y ventas, se nos dieron varios requerimientos a cumplir. Comenzamos haciendo el analisis de requerimientos para poder hacer el diseño de nuestra base de datos, una vez teniendo el diseño de la base de datos será necesario hacer la implementacion de funciones que den solución a los requerimientos solicitados, ya teniendo toda la parte del funcionamiento el siguiente objetivo será la implementación de la interfaz grafica.

2 Plan de Trabajo

Para realizar este proyecto debiamos tener una planeación de tiempos muy precisa ya que el tiempo de desarrollo no era demasiado, durante las primeras juntas asignamos roles de trabajo, fechas tentativas y definimos cual podría ser la solución al problema y requerimientos solicitados, a continuación adjuntamos un cronograma de trabajo en el cual se mencionan las fechas y actividades que realizamos.

1. MER
2. MR
3. Creacion de una base de datos junto con la creacion de las tablas
4. La creacion de las funciones junto a sus triggers
5. La creacion del Front End o una interfaz grafica
6. Y por ultimo pero no menos importante la conexion de la BD con el Front End

3 Diseño

Para comenzar con la fase de diseño es necesario haber hecho un análisis detallado del problema para así poder brindar una solución óptima para este. Trabajamos tal cual lo hacíamos en clase así que el primer paso era encontrar cuales son las *Entidades* del problema, las cuales mencionaremos a continuación:

- Provedor
- Poducto
- Inventario
- Venta
- Cliente

Una vez identificadas las entidades procedimos a analizar y asignar cada uno de sus atributos y las relaciones entre ellas para así poder hacer el diagrama MER y el MR los cuales mostramos a continuación

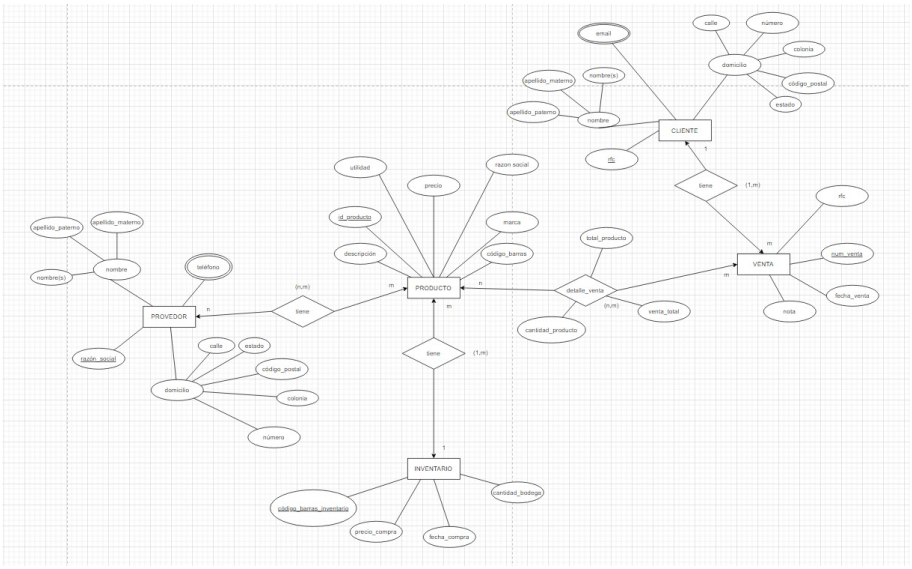


Diagrama del MER

Si no alcanzar a ver bien el diagrama, puedes acceder dandole click aqui :D

Gracias a ese diagrama podemos plantearnos como vamos a hacer el **MR**, aunque tampoco fue complicardo hacerlo se pudo lograr terminarlo y quedando de la siguiente manera

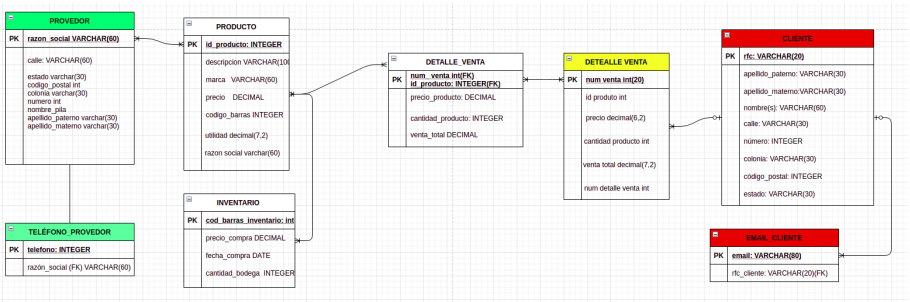


Diagrama MR

Teniendo todo eso ya echo nos da una idea de como podemos hacer la **"BASE DE DATOS"** que es la parte mas emocionante de todo donde se delimitan donde estan los niños con los hombres y los veremos en el siguiente punto.

4 Implementación

Esta fase es en la que ponemos en práctica todo el análisis previamente realizado en las demás fases, comenzamos creando la base de datos que usaremos en nuestro proyecto, ahora hay que crear las tablas que conformaran esta base de datos, previamente en el MR definimos el tipo de dato de cada atributo, el MER y MR nos facilitan este proceso de la creación de las tablas. Para este proceso de la implementación usamos lenguaje sql, podríamos hacer la implementación desde la terminal o desde un manejador incluso la podríamos hacer en un editor de código que usaremos para hacer el frontend, del cual hablaremos más adelante. A continuación podemos ver el código que utilizamos para creas la base de datos y las tablas que la conforman

```
CREATE DATABASE Shop;
-- Tabla proveedor
CREATE TABLE PROVEDOR(
razon_social varchar(60) NOT NULL,
calle varchar(30) NOT NULL,
estado varchar(30) NOT NULL,
codigo_postal int NOT NULL,
colonia varchar(30) NOT NULL,
numero int NOT NULL check(numero>0),
nombre_pila VARCHAR(30) NOT NULL,
apellido_paterno VARCHAR(30) NOT NULL,
apellido_materno VARCHAR(30)  NULL,
CONSTRAINT razon_social_PK PRIMARY KEY (razon_social)
```

```
);

-- Tabla teléfonos
CREATE TABLE TELEFONO_PROVEDOR(
telefono int NOT NULL ,
razon_social varchar(60) NOT NULL,
CONSTRAINT TELEFONO_PK PRIMARY KEY(razon_social,telefono),
CONSTRAINT PROVEEDOR_FK FOREIGN KEY (razon_social) REFERENCES
PROVEDOR(razon_social)
);

-- Tabla inventario
CREATE TABLE INVENTARIO(
codigo_barras_inventario int NOT NULL,
precio_compra DECIMAL(6,2)  NOT NULL CHECK (precio_compra>0),
fecha_compra date NOT NULL default now(),
cantidad_bodega int NOT NULL CHECK(cantidad_bodega>=0),
CONSTRAINT codigo_barras_PK PRIMARY KEY (codigo_barras_inventario)
);

-- Tabla producto
CREATE TABLE PRODUCTO (
id_producto int  GENERATED ALWAYS AS IDENTITY NOT NULL,
descripcion varchar(100)  NULL,
marca VARCHAR(60) NOT NULL,
precio DECIMAL(6,2) NOT NULL CHECK(precio>0),
codigo_barras int NOT NULL,
utilidad DECIMAL(7,2) NOT NULL ,
razon_social varchar(60) NOT NULL,
CONSTRAINT id_producto_PK PRIMARY KEY (id_producto),
CONSTRAINT PROVEEDOR_PRODUCTO_FK FOREIGN KEY (razon_social)
REFERENCES PROVEDOR (razon_social),
CONSTRAINT INVENTARIO_FK FOREIGN KEY (codigo_barras)
REFERENCES INVENTARIO(codigo_barras_inventario) ON DELETE CASCADE
);

-- Tabla Cliente
CREATE TABLE CLIENTE(
 rfc varchar(20) NOT NULL,
 apellido_paterno VARCHAR(30) NOT NULL,
 apellido_materno VARCHAR(30)  NULL,
 nombre VARCHAR(60) NOT NULL,
 calle varchar(30) NOT NULL,
 numero int CHECK (numero>0),
 colonia varchar(30) NOT NULL,
 codigo_postal int CHECK (codigo_postal>0) NOT NULL,
```

```
 estado VARCHAR(30) NOT NULL,
 CONSTRAINT razon_cliente_PK PRIMARY KEY (rfc)
);

-- Tabla email del cliente
CREATE TABLE EMAIL(
email varchar(80) NOT NULL,
rfc_cliente varchar(20) NOT NULL,
CONSTRAINT EMAIL_PK PRIMARY KEY(rfc_cliente,email),
CONSTRAINT CLIENTE_FK FOREIGN KEY (rfc_cliente)
REFERENCES CLIENTE (rfc)
);

-- Tabla venta
CREATE TABLE VENTA(
 num_venta int GENERATED ALWAYS AS IDENTITY  NOT NULL,
 fecha_venta date NOT NULL,
 nota VARCHAR(15) null default 'VENT-',
 rfc_cliente varchar(20) NOT NULL,
 CONSTRAINT VENTA_PK PRIMARY KEY (num_venta),
 CONSTRAINT VENTA_CLIENTE_FK FOREIGN KEY (rfc_cliente)
 REFERENCES CLIENTE(rfc)
);

-- Tabla detalle venta
CREATE TABLE DETALLE_VENTA(
num_venta int  NOT NULL,
id_producto int  NOT NULL,
precio_producto DECIMAL(6,2) not null,
cantidad_producto int NOT NULL,
venta_total DECIMAL(7,2) NOT NULL,
num_detalle_venta int GENERATED ALWAYS AS IDENTITY not null,
CONSTRAINT DETALLE_VENTA_PK PRIMARY KEY (num_venta,id_producto),
CONSTRAINT ORDEN_PRODUCTO_FK FOREIGN KEY (id_producto)
REFERENCES  PRODUCTO(id_producto) on delete CASCADE,
CONSTRAINT VENTA_FK FOREIGN KEY (num_venta)
REFERENCES  VENTA(num_venta)on delete CASCADE
);
```

Todas las tablas vistas anteriormente estan creadas y pensadas para dar solución a varios requerimientos que se solicitó resolver con el uso de funciones, ademas de funciones implementamos triggers los cuales se ejecutaran automaticamente en el momento que se ejecuten las funciones, todas estas fueron hechas como solución a los requerimientos del proyecto

FUNCIONES

```
CREATE OR REPLACE FUNCTION FUNCION_CODIGO_BARRAS(codigo int)
```

```
RETURNS SETOF PRODUCTO AS $$
BEGIN
RETURN QUERY SELECT * FROM PRODUCTO where codigo_barras=codigo;
END
$$ LANGUAGE plpgsql;
```

La función se usa para obtener la información sobre el producto dado un determinado código de barras

```
CREATE OR REPLACE FUNCTION reinicia_id_producto()
RETURNS void AS $$
declare id_start int;
BEGIN
begin
select max(id_producto)+1 from PRODUCTO into id_start;
execute 'alter SEQUENCE producto_id_producto_seq  RESTART with '|| 1;
end;
END
$$ LANGUAGE plpgsql;
```

En esta función reinicia el contador de id producto

```
CREATE OR REPLACE FUNCTION verifica_codigo_repetido()
RETURNS trigger AS
$$
declare id_funcion int;
begin
if exists((SELECT codigo_barras,razon_social FROM
PRODUCTO group by codigo_barras,razon_social having count(*)>1))then
raise notice 'Error al insertar: Ya hay un producto con
el mismo codigo de barras y proveedor';
select (max(id_producto)-1) from PRODUCTO
into id_funcion;
execute 'alter SEQUENCE ciclo_producto_id
RESTART with '|| id_funcion;
delete from PRODUCTO where id_producto=(select (max(id_producto))
from PRODUCTO);
return new;
else
select max(id_producto)+1 from PRODUCTO into id_funcion;
execute 'alter SEQUENCE ciclo_producto_id RESTART with'|| id_funcion;
UPDATE PRODUCTO
set utilidad=(select precio from PRODUCTO group by
precio having max(id_producto)=(select max(id_producto) from PRODUCTO))
-(select precio_compra from INVENTARIO
where codigo_barras=(select codigo_barras from PRODUCTO
group by codigo_barras having max(id_producto)=
```

```
(select max(id_producto) from PRODUCTO)))
where id_producto=(select max(id_producto) from PRODUCTO);
raise notice 'Inserción exitosa';
return new;
end if;
END;
$$
LANGUAGE plpgsql;
```

En esta función se revisa que no haya ningun codigo de barras repetido si es así entonces lo elimina para que no existan valores repetidos

```
CREATE TRIGGER trigger_borrar_detalle_venta
after delete
ON DETALLE_VENTA
FOR EACH ROW
EXECUTE PROCEDURE funcion_verificar_borrado_detalle_venta();
```

Trigger que es llamado después de que se elimina algún dato de la tabla DETALLE-VENTA para restablecer la secuencia para cada venta ya que se crearon con la propiedad IDENTITY.

```
CREATE OR REPLACE FUNCTION funcion_verificar_borrado_detalle_venta()
RETURNS trigger AS
$$
declare variable_funcion int;
begin
if(exists(select num_venta from DETALLE_VENTA)) then
select (max(num_venta)+1) from DETALLE_VENTA into variable_funcion;
execute 'alter SEQUENCE detalle_venta_numero  RESTART with '||
variable_funcion;
return new;
else
execute 'alter SEQUENCE detalle_venta_numero RESTART with '|| 1;
return new;
end if;
END;
$$
LANGUAGE plpgsql;
```

Esta función controla el orden de los detalles de la venta ya que al ser creadas con identity y si se incerta una nueva venta la secuencia seguirá aumentando aún cuando el elemento haya sido borrado. Por lo tanto fue necesario crear esta función para que al momento de eliminar el detalle de alguna venta no nos arroje futuros errores.

```
CREATE OR REPLACE FUNCTION vista_informacion_por_orden
(No_orden_cliente_recibida varchar(8))
```

```
RETURNS  table(
               num_venta int,
               fecha date,
               Nombre varchar(60),
               cantidad_producto decimal(6,2),
               venta_total DECIMAL(7,2)) AS
$$
begin
return query (select dv.num_venta,dv.fecha_venta,concat(
c.nombre_pila,' ',c.apellido_p,' ',c.apellido_m),
    count(v.cantidad_articulo),SUM(v.total_pagar)
    from DETALLE_VENTA as dv inner join
    VENTA  as v on dv.num_venta=v.num_venta
    inner join CLIENTE as c on c.razon_cliente=dv.razon_cliente
    where dv.nota_venta=No_orden_cliente_recibida group by
    dv.No_venta,dv.fecha_venta,c.nombre_pila,c.apellido_p,c.apellido_m);
END;
$$
LANGUAGE plpgsql;
```

La función regresa una nueva tabla compuesta del número de la venta, la fecha, el nombre del cliente, la cantidad del producto y el total de la venta. De esta forma podemos tener a la mano la información más importante e indispensable de una venta.

```
    CREATE OR REPLACE FUNCTION stock_menor_3()
RETURNS  table(nombre  varchar(40))
as
$$
begin
return query (select p.nombre from  INVENTARIO i inner join
prvucto p on i.cvigo_barras=p.cvigo_barras where i.unidades_stock<3);
END;
$$
LANGUAGE plpgsql;
```

Esta función sirve para obtener los productos cuya cantidad en el inventario es menor a tres unidades. De esta forma podemos llevar un registro de los productos necesita la tienda abastecerse.

```
CREATE OR REPLACE FUNCTION venta_ticket(orden_del_cliente VARCHAR(8))
RETURNS  table(
    num_venta int,
    fecha_venta date,
    nombre VARCHAR(60),
    descripcion varchar(100),
```

```
    marca varchar(60),
    cantidad_producto int,
    precio_producto decimal(6,2),
    venta_total decimal(7,2)) AS
$$
begin
return query (select v.nota,v.fecha_venta,concat
(c.nombre,' ',c.apellido_paterno,' ',c.apellido_materno),
    p.descripcion,p.marca,dv.cantidad_producto,
    dv.precio_producto,dv.venta_total
    from VENTA as v inner join
    DETALLE_VENTA  as dv on v.num_venta=dv.num_venta
    inner join CLIENTE as c on c.razon_cliente=v.razon_cliente
    inner join PRODUCTO as p on p.id_producto=dv.id_producto
    where v.nota=orden_del_cliente);
END;
$$
LANGUAGE plpgsql;
```

Regrese un "ticket". Éste es en realidad una nueva tabla con información de 3 tablas: venta,detalle-venta,cliente y producto. Esta función esta implementada para que funcione bastante parecido a un ticket de compra.

```
CREATE OR REPLACE FUNCTION crea_num_venta()
RETURNS trigger AS
$$
begin
if(exists(select num_venta from DETALLE_VENTA)) then
new.nota=concat(new.nota,(RIGHT((concat(
'000' , CAST(new.num_venta AS VARCHAR(3)))),3)));
return new;
else
new.nota=concat(new.nota,(RIGHT((concat
('000' , CAST(new.num_venta AS VARCHAR(3)))),3)));
return new;
end if;
END;
$$
LANGUAGE plpgsql;
```

Función que crea el formato que solicitado para una venta: VENT-000. De esta forma se tiene un mejor control de todas las ventas del negocio.

```
CREATE TRIGGER trigger_num_venta
```

```
before insert
ON DETALLE_VENTA
FOR EACH ROW
EXECUTE PROCEDURE crea_num_venta();
```

Trigger que activa la función anterior para que cada vez que se crea una venta le asigne el formato pedido.

```
CREATE OR REPLACE FUNCTION verifica_codigo_test()
RETURNS trigger AS
$$
begin
if(exists(select id_producto from PRODUCTO)) then
return new;

else
return new;
end if;

END;
$$
LANGUAGE plpgsql;
```

Verifica mediante el ID del producto si existe el producto solicitado mediante una consulta a su ID. Esta funcion fue creada para realizar pruebas

```
CREATE TRIGGER verifica_trigger
before insert
ON PRODUCTO
FOR EACH ROW
EXECUTE PROCEDURE verifica_codigo_test();
```

Trigger para verificar el código de un producto.

```
CREATE OR REPLACE FUNCTION cantidad_total_venta
(fecha_inicio DATE, fecha_fin DATE)
RETURNS  table(cantidad_vendida  numeric)
as
$$
begin
return query (select sum(dv.vental_total)
              from VENTA as ve inner join
              DETALLE_VENTA  as dv on ve.num_venta=dv.num_venta
              where ve.num_venta=dv.num_venta and ve.fecha_venta
              between fecha_inicio and fecha_fin);
END;
$$
LANGUAGE plpgsql;
```

Función que regresa la cantidad vendida dadas dos fechas, una de inicio y una final.

```
    CREATE OR REPLACE FUNCTION multifuncion_stock()
RETURNS trigger AS
$$
declare id_funcion int;
begin
-- if para saber si hay suficientes unidades parahacerel insert
-- auqnue el insert ya sehizo
if((select cantidad_bodega - (select cantidad_articulo from
detalle_venta where num_venta=(select num_venta from
detalle_venta  group by num_venta
      having max(num_venta)=(select max(num_venta) from
      detalle_venta )))from INVENTARIO where codigo_barras=(
      select codigo_barras from PRODUCTO P where id_producto=
        (select id_producto from detalle_venta where num_venta=(
        select num_venta from detalle_venta  group by num_venta
           having max(num_venta)=(select max(num_venta) from
           detalle_venta )))))< 0) then raise notice 'Inventario
           lleno: Venta no registrada';
            delete from detalle_venta where num_venta=(select (
            max(num_venta)) from detalle_venta);
             return null;
else
--si hay inventario suficientehacemos el update  de campo
precio_por_unidad del producto
UPDATE detalle_venta
 set precio_producto=
  (select precio_venta from PRODUCTO P where id_producto=(
  select id_producto from detalle_venta where num_venta=(
  select num_venta from detalle_venta group by num_venta
    having max(num_venta)=(select max(num_venta) from
    detalle_venta )))) where num_venta=(select max(num_venta)
    from detalle_venta);
-- actualizamos el total a pagar
    UPDATE detalle_venta set total_pagar=(select cantidad_articulo*(
    select precio_producto from detalle_venta where num_venta=(
    select num_venta from detalle_venta  group by num_venta having max(
      num_venta)=(select max(num_venta) from detalle_venta )))
      from detalle_venta
       where num_venta=(select num_venta from detalle_venta
       group by num_venta having max(num_venta)=(select max(
       num_venta) from detalle_venta )))
        where num_venta=(select max(num_venta) from detalle_venta );
-- actualizamos el inventario
```

```
     UPDATE INVENTARIO set cantidad_bodega= (select cantidad_bodega -(
     select cantidad_articulo from detalle_venta where num_venta=(
     select num_venta from detalle_venta  group by num_venta
      having max(num_venta)=(select max(num_venta) from detalle_venta )))
      from INVENTARIO where codigo_barras=(select codigo_barras from
      PRODUCTO P where id_producto=
        (select id_producto from detalle_venta where num_venta=(select
        num_venta from detalle_venta  group by num_venta having max(
        num_venta)=(select max(num_venta) from detalle_venta )))))
        where codigo_barras=(select codigo_barras from PRODUCTO P
        where id_producto=(select id_producto from detalle_venta
            where num_venta=(select num_venta from detalle_venta
            group by num_venta having max(num_venta)=(select max(
            num_venta) from detalle_venta ))));
--si quedan menor que 3 en el inventario
if((select cantidad_bodega -(select cantidad_articulo from
detalle_venta where num_venta=(select num_venta from detalle_venta
group by num_venta
      having max(num_venta)=(select max(num_venta) from detalle_venta )))
      from INVENTARIO where codigo_barras=(select codigo_barras from
      PRODUCTO P where id_producto=(select id_producto from detalle_venta
      where num_venta=(select num_venta from detalle_venta  group by
      num_venta having max(num_venta)=(select max(num_venta)
      from detalle_venta )))))<=3) then
          raise notice 'Advertencia: Stock del producto es menor o igual a 3';
end if;
 raise notice 'No se pudo insertar';
     return new;
 end if;
END;
$$
LANGUAGE plpgsql;
```

Función que realiza varias tareas: revisa si se puede hacer el insert, se asegura de que podamos hacer un update, después de eso actualizamos el inventario y el total a pagar. Además revisa si algún producto cuenta con un stock en el inventario de menos de 3 unidades, de ser asi manda un mensaje de aviso. Todas las funcionalidades anteriores se evalúan después de realizar una compra.

```
CREATE OR REPLACE FUNCTION rev_borrado_producto()
RETURNS trigger AS
$$
declare id_function int;
begin
if exists(select id_producto from PRODUCTO) then
select (max(id_producto)+1) from PRODUCTO into id_function;
```

```
execute 'alter SEQUENCE producto_id_producto_seq  RESTART with '|| id_function;
return new;
else
perform reinicia_id_producto();
return new;
end if;
END;
$$
LANGUAGE plpgsql;
```

Esta función revisa que el producto que queramos borrar exista y también en caso de que si exista restablece el ID para que no se produzcan errores al momento de insertar nuevos productos devido al IDENTITY.

```
CREATE TRIGGER trigger_borrado_actualiza_id
after delete
ON PRODUCTO
FOR EACH ROW
EXECUTE PROCEDURE rev_borrado_producto();
```

Trigger que llama a la función anterior para que cada vez que se borra un producto se actualize la secuencia del IDENTITY.

```
CREATE OR REPLACE FUNCTION verifica_borrado_venta()
RETURNS trigger AS
$$
declare max_id int;
begin
if(exists(select (No_venta) from VENTA)) then
select (max(num_venta)+1) from VENTA into max_id;
execute 'alter SEQUENCE venta_num_venta_seq  RESTART with '|| max_id;
return new;
else
execute 'alter SEQUENCE venta_num_venta_seq  RESTART with '|| 1;
return new;
end if;
END;
$$
LANGUAGE plpgsql;
```

Similar a la función anterior, verifica que exista una venta determinada anter de poder borrarla y también en caso de que sí exista restablece el ID para que no se produzcan errores al momento de insertar nuevos productos devido al IDENTITY.

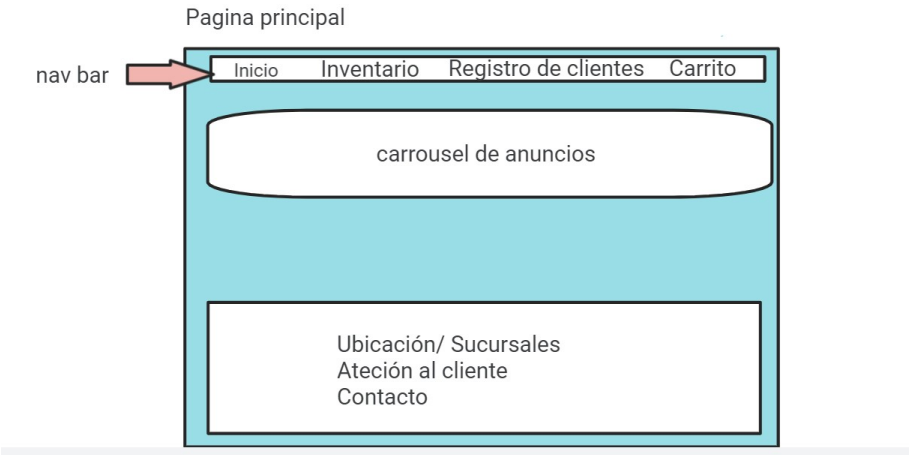
```
CREATE TRIGGER trigger_borrado_venta
```

```
after delete
ON VENTA
FOR EACH ROW
EXECUTE PROCEDURE verifica_borrado_venta();
```

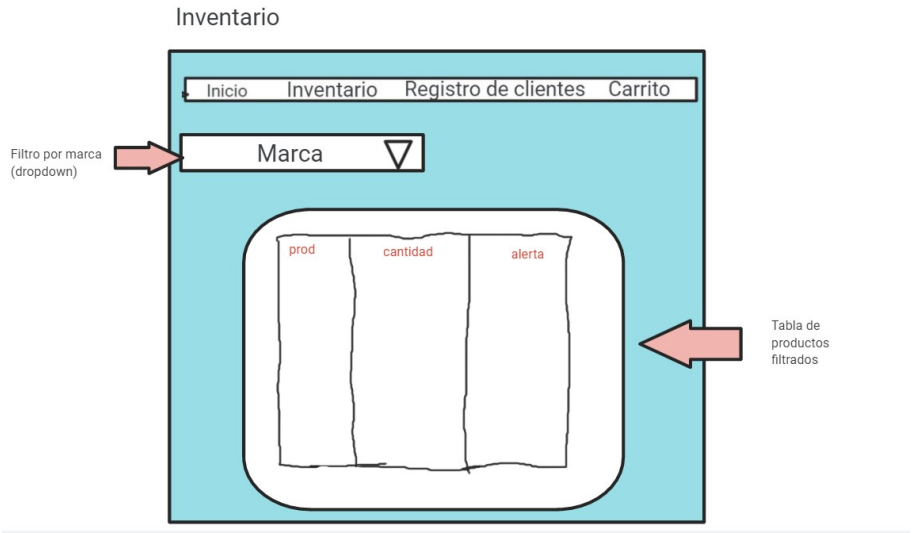
Trigger que llama a la función anterior para que cada vez que se borra una venta se actualize la secuencia del IDENTITY.

5 Presentación

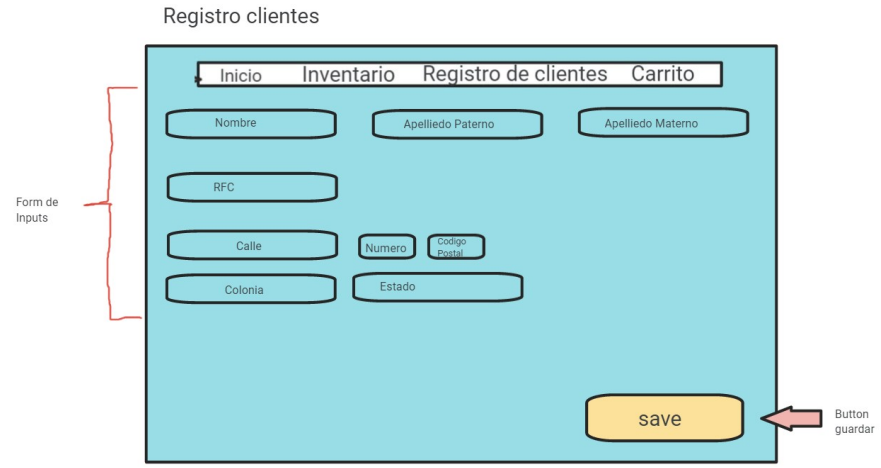
Entendemos que en esta sección se presentara el procedimiento que hicimos para la parte del frontend, lo primero como en la parte de backend es hacer un análisis detallado del requerimiento para así poder cubrir todo lo solicitado, en este análisis decidimos separar en modulos, una vez establecidos los modulos procedemos a ver le flujo de nuestra pagina web y a maquetar, maquetamos los diseños de la interfaz para ver que esta cumpla con lo necesario. Una vez hecho esto decidimos el lungaje que se usara, el cual es JavaScript A contuniación dejamos las maquetas que hicimos para las pantallas (se mando un correo al profesor en el cual se nos autorizo subir la parte faltante del frontend antes de nuestra presentación, razón por la cual no adjuntamos el código empleado)



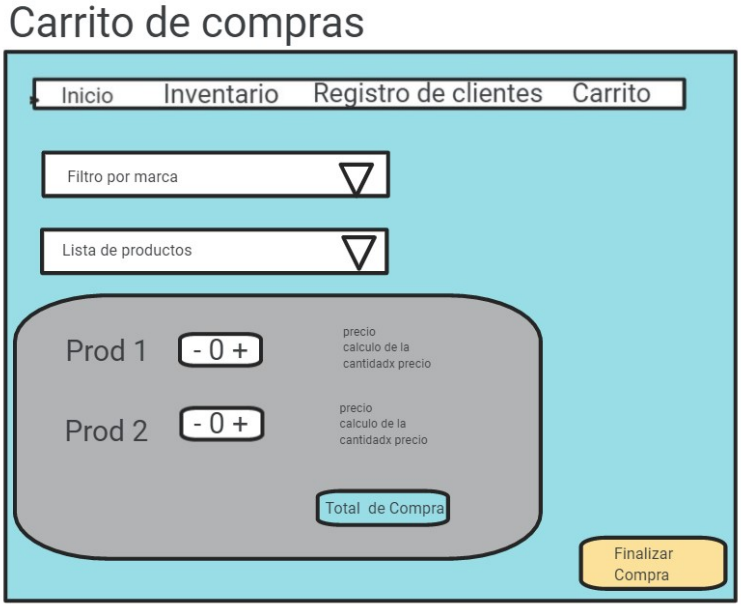
Maqueta Pantalla Principal



Maqueta Pantalla Inventario



Maqueta Pantalla Clientes



Maqueta Pantalla Carrito de Compras

6 Conclusiones

Camacho Lara Ana Laura
Me parece que el proyecto fue un buen reto para nosotros ya que se parece bastante a todo lo que se debe hacer en el mundo laboral, es bueno practicarlo desde ahora. En cuanto el desarrollo de este pude poner en practica absolutamente todo lo que vimos en clase y darme cuenta que si había aprendido y que temas me falto reforzar un poco lo cual pude hacer mientras desarrollabamos el protecto. Este estuvo lleno de retos como la implementación de las funciones y triggers y posteriormente hacer el frontend, en este punto lo que más se me complico fue entender como ocupar los datos que traemos de la base de datos al frontend para poder mostrarlas con el diseño que habíamos elegido.

Martínez Matías Joan Eduardo
El proyecto fue uno de los mas complejos y completos que tuve éste semestre, ya que realizamos un analisis muy completo para la realización de este, tuvimos que hacer una base de datos desde cero, ademas de los modelos MR y MER que fueron las bases para empezar nuestra base de datos. Además sobre la marcha ibamos realizando nuevas tareas y editando algunas para tener un mejor resultado de nuestro proyecto. Lo que mas me agradó fue aplicar todo lo aprendido en la materia y poderlo aplicar a una pagina web que es todo un reto hacerla, ademas de investigar como poder conectar la pagina con la base de datos. Me

llevo mucho aprendizaje de este trabajo, tambien un aprendizaje de trabajar en equipo y colaborativamente.

Vázquez Torres Juan Adrián

Este proyecto fue uno de los más completos que he realizado dentro de la universidad, porque tuvimos que crear una base de datos desde 0, es decir: análisis de requerimientos, construcción del MER, representación intermedia, construcción del MR, creación de tablas, creación de funciones y creación de triggers. También tuvimos (por voluntad) que construir una página web que da soporte a nuestro negocio (papelería) y tuvimos que conectarla a la base de datos, por lo que tuvimos que adentrarnos en herramientas externas a la DB como javascript, React, express, etc. Fue muy interesante el proyecto aunque no fue fácil debido a la modalidad en línea y a diversas situaciones que se presentaron en nuestra facultad.

Zetina Muciño Aldo.

El programa fue todo un reto porque es crear una base de datos desde cero con los conocimiento que le profesor nos brindo con **PostgreSQL** si fue divertido aprender esto y me gusto muchisimo al mometno de hacerlo pero claro no fue facil, tener que desvelarme a las 2 de la mañana para hacer el programa el codigo SQL investigar y saber que es lo que estoy haciendo y porque, un que siento que lo mas dificil es hacer la coneccion del front end con la base de datos porque son temas un poco mas avanzados que no nos corresponde a nosotros en este temario pero igualmente se vio como un reto y aparte para llevarnos un aprendisaje de esto. :)