Proyecto de bases de datos
Papeletta

Martínez Rojas Jose Eduardo
Vázquez Torres Juan Adrián
Báez Cadena Diestefano Michel

Fecha entrega: 25 enero de 2021

1 Objetivo

Diseñar una base datos para una cadena de papelerías que busca innovar la manera en que almacena su información.

2 Introducción

Papeleta una consultora se encargará de diseñar una solución adecuada y simple para este proyecto, se propondrá y diseñerá primero una solución para la base de datos y terminando, se empezará la interfez gráfica. Para este proyecto se tomarán los requisitos de la cadena de papelerías, tanto en diseño como en desarrollo e implementación de la base datos, en cuanto a la interfaz se seguirá el diseño ya sea una app móvil o una página web.

3 Análisis del Problema

El proyecto consiste en la elaboración de una base de datos para una cadena de papelarías que quiere adaptarse a esta epoca tecnológica. El cliente solicitó que la base de datos almacenara la información de los proveedores, de los clientes, el inventario de los productos con su respectiva información, el stock en almacén y las ventas que se hacen. Cada uno de estos puntos está compuesto por información detallada, la cual tiene que ser almacenada para su posterior consulta y manejo.

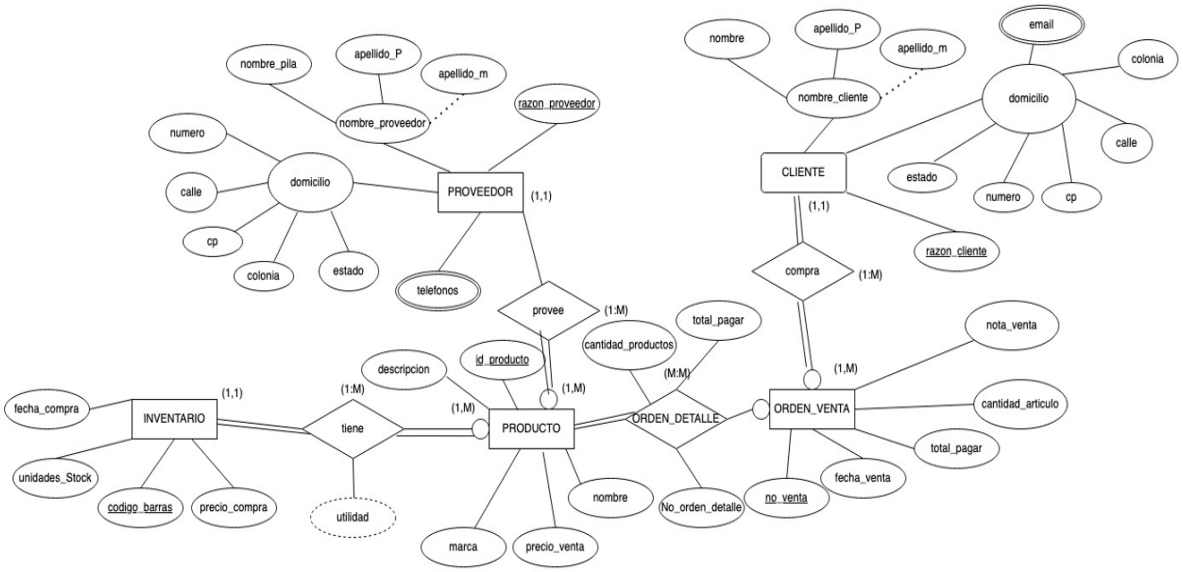
4 Diseño

Una vez consultado todos los requerimientos solicitados por el cliente, se obtuvieron los siguientes datos. Se desea tener almacenados datos como la razón social, domicilio, nombre y teléfonos de los proveedores, razón social, nombre, domicilio y al menos un email de los clientes. Es necesario tener un inventario de los productos que se venden, en el que debe guardarse el código de barras, precio al que fue comprado el producto, fecha de compra y cantidad de ejemplares en la bodega (stock). Se desea guardar la marca, descripción y precio de los regalos, artículos de papelería, impresiones y recargas, siempre y cuando se tenga su correspondiente registro en el inventario. Debe también guardarse el número de venta, fecha de venta y la cantidad total a pagar de la venta, así como la cantidad de cada artículo y precio total a pagar por artículo. Además, se requiere que:

- Al recibir el código de barras de un producto, regrese la utilidad.
- Cada que haya la venta de un artículo, deberá decrementarse el stock por la cantidad vendida de ese artículo. Si el valor llega a cero, abortar la transacción. Si hay menos de 3, emitir un mensaje.
- Dada una fecha, o una fecha de inicio y fecha de fin, regresar la cantidad total que se vendió en esa fecha/periodo.
- Permitir obtener el nombre de aquellos productos de los cuales hay menos de 3 en stock.
- De manera automática se genere una vista que contenga información necesaria para asemejarse a una factura de una compra.
- Crear al menos, un índice, del tipo que se prefiera y donde se prefiera.
- El número de venta debe tener un formato similar a "VENT-001", prefijo VENT, seguido de un guión y un número secuencial.
- Donde este presente el atributo domicilio, está compuesto por estado, código postal, colonia, calle y número. El diseño debe satisfacer todos los principios de diseño, los requerimientos anteriores y un buen manejo de información.

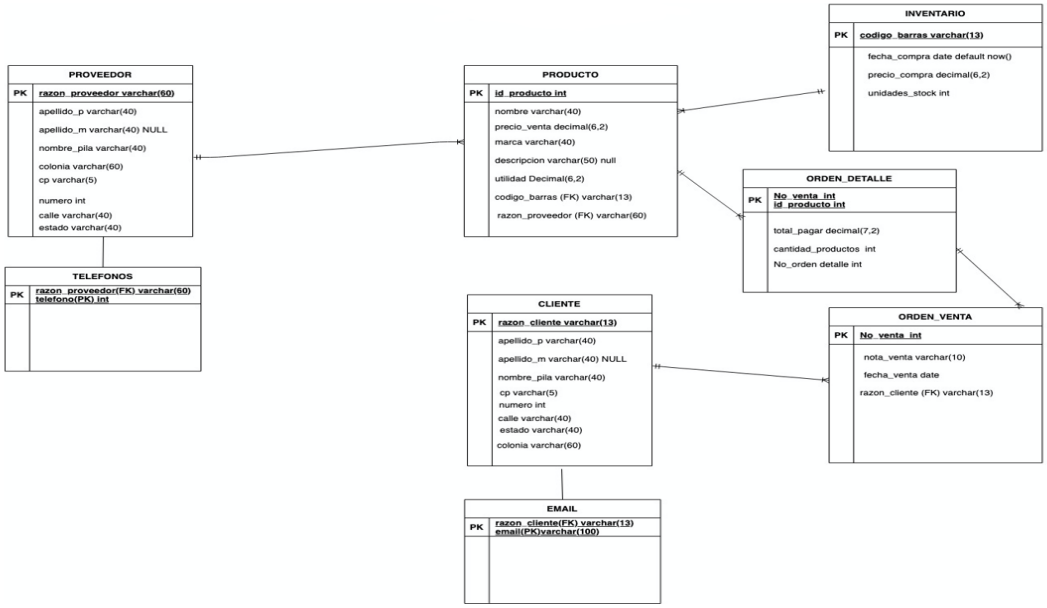
Modelo Entidas Relación

En la elaboraicón del modelo entidad relación se identificaron las entidades de la base de datos a implementar, también se sacaron los atributos que le corresponde a cada identidad, las relaciones entre entidades y su cardinalidad. Esta etapa del diseño es muy importante porque si no se establecen bien las entidades junto con sus atributos y las relaciones, puede causar muchos problemas en etapas posteriores, por el contrario, si se hace bien podemos ahorrar tiempo y recursos en el proyecto.



Modelo Relacional

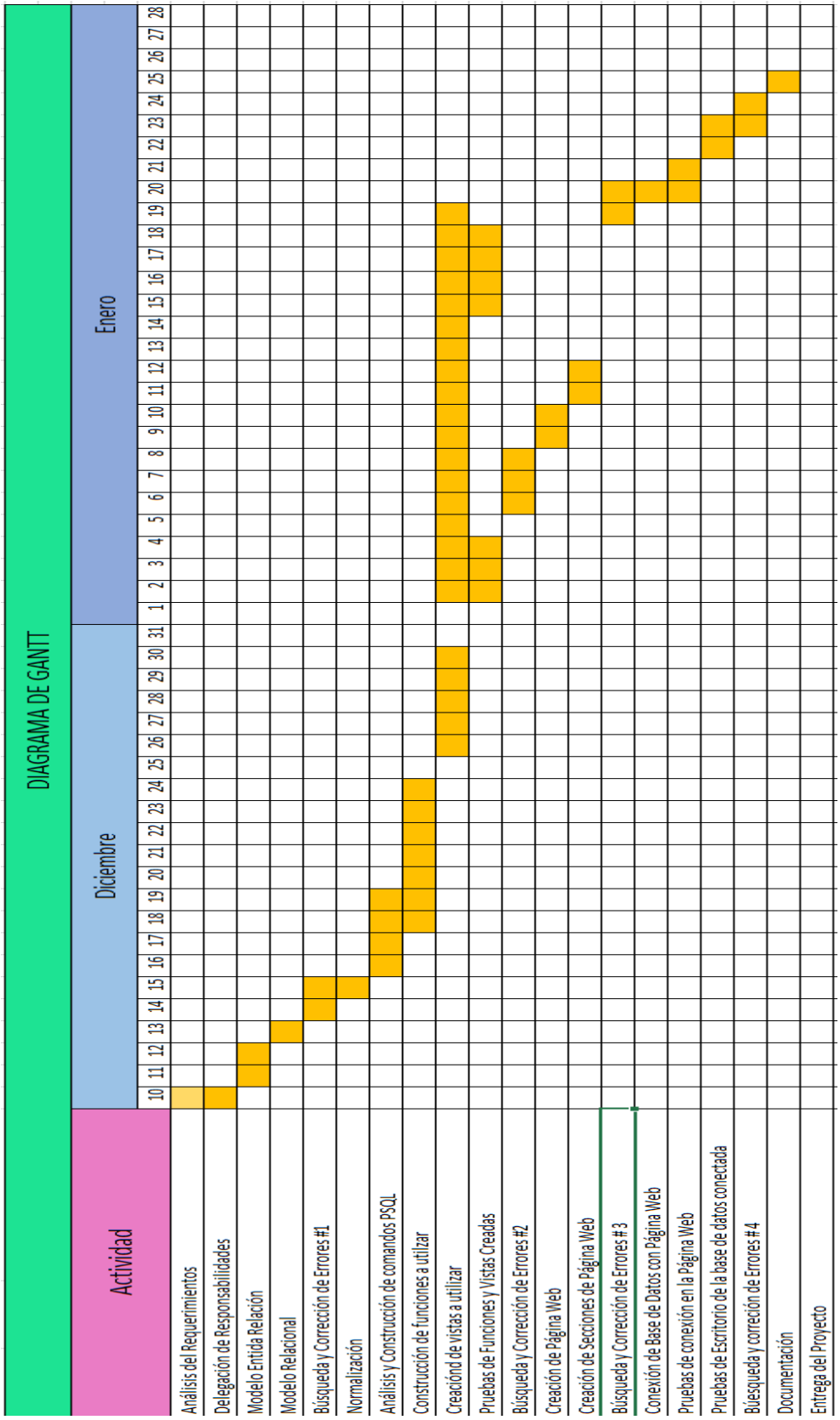
Una vez que se revisó el modelo entidad relación y se aseguró que estuviera bien elaborado el siguiente paso en el proceso de diseño fue la elaboración del modelo relacional. En esta parte de la construcción del proyecto se determinaron los tipos de datos de los atributos junto con su extensión. Además se volvieron a revisar las llaves primarias de las entidades para asegurarse de que fueron elegidas correctamente y no causen problemas en el futuro.



Construcción de Comandos PgSQL

La construcción de los comandos fue realizada de acuerdo con lo visto en la asignatura (bases de datos relacionales), primero se analizó el tipo de estructura y lógica adecuada para cada requerimiento de la base de datos, después se codificaron las instrucciones y finalmente se crearon los triggers, vistas, funciones y procedures necesarios para cada requerimiento.

5 Plan de Trabajo



6 Implementación de Código

6.1 Tablas

Tabla PROVEEDOR

```
CREATE TABLE PROVEEDOR(
                        razon_proveedor varchar(60) NOT NULL,
                        apellido_p VARCHAR(40) NOT NULL,
                        apellido_m VARCHAR(40)  NULL,
                        nombre_pila VARCHAR(60) NOT NULL,
                        colonia varchar(40) NOT NULL,
                        estado varchar(40) NOT NULL,
                        calle varchar(40) NOT NULL,
                        cp varchar(5) NOT NULL,
                        numero int NOT NULL check(numero>0),
                        CONSTRAINT razon_proveedor_PK PRIMARY KEY (razon_proveedor));
```

Tabla TELÉFONOS

```
CREATE TABLE TELEFONOS(
                    razon_proveedor varchar(60) NOT NULL,
                    telefono varchar(10) NOT NULL ,
                    CONSTRAINT TELEFONO_PK PRIMARY KEY(razon_proveedor,telefono),
                    CONSTRAINT PROVEEDOR_FK FOREIGN KEY (razon_proveedor) REFERENCES  PROVEEDOR (razon_proveedor)
                    );
```

Tabla INVENTARIO

```
CREATE TABLE INVENTARIO(
                        codigo_barras varchar(13) NOT NULL ,
                        fecha_compra date NOT NULL default now(),
                        precio_compra DECIMAL(6,2)  NOT NULL CHECK (precio_compra>0),
                        unidades_stock int NOT NULL CHECK(unidades_stock>=0) ,
                        CONSTRAINT codigo_barras_PK PRIMARY KEY (codigo_barras)
                        );
```

Tabla PRODUCTO

```
CREATE TABLE PRODUCTO (
                    id_producto int  GENERATED ALWAYS AS IDENTITY NOT NULL,
                    nombre VARCHAR(40) NOT NULL,
                    precio_venta DECIMAL(6,2) NOT NULL CHECK(precio_venta>0),
                    marca VARCHAR(40) NOT NULL,
                    descripcion varchar(50)  NULL,
                    codigo_barras varchar(13) NOT NULL,
                    utilidad DECIMAL(6,2) NOT NULL ,
                    razon_proveedor varchar(60) NOT NULL,
                    CONSTRAINT id_producto_PK PRIMARY KEY (id_producto),
                    CONSTRAINT INVENTARIO_FK FOREIGN KEY (codigo_barras) REFERENCES INVENTARIO(codigo_barras) ON DELETE CASCADE,
                    CONSTRAINT PROVEEDOR_PRODUCTO_FK FOREIGN KEY (razon_proveedor) REFERENCES PROVEEDOR (razon_proveedor)
                    );
```

Tabla CLIENTE

```
CREATE TABLE CLIENTE(
                        razon_cliente varchar(13) NOT NULL,
                        apellido_p VARCHAR(40) NOT NULL,
                        apellido_m VARCHAR(40)  NULL,
                        nombre_pila VARCHAR(60) NOT NULL,
                        colonia varchar(40) NOT NULL,
                        estado varchar(40) NOT NULL,
                        calle varchar(40) NOT NULL,
                        cp varchar(5) NOT NULL,
                        numero int CHECK (numero>0),
                        CONSTRAINT razon_cliente_PK PRIMARY KEY (razon_cliente)
                        );
```

Tabla ORDEN_VENTA

```
CREATE TABLE ORDEN_VENTA(
                        No_venta int GENERATED ALWAYS AS IDENTITY  NOT NULL,
                        fecha_venta date NOT NULL,
                        razon_cliente varchar(13) NOT NULL,
                        nota_venta VARCHAR(10) null default 'VENT-',
                        CONSTRAINT ORDEN_VENTA_PK PRIMARY KEY (No_venta),
                        CONSTRAINT CLIENTE_FK_VENTA FOREIGN KEY (razon_cliente) REFERENCES CLIENTE(razon_cliente)
                        );
```

Tabla ORDEN_DETALLE

```
CREATE TABLE ORDEN_DETALLE(
                        No_orden_detalle int GENERATED ALWAYS AS IDENTITY not null,
                        No_venta int  NOT NULL,
                        id_producto int  NOT NULL,
                        cantidad_articulo int NOT NULL,
                        precio_venta_producto DECIMAL(6,2) not null,
                        total_pagar DECIMAL(7,2) NOT NULL,
                        CONSTRAINT ORDEN_VENTA__DETALLE_PK PRIMARY KEY (No_venta,id_producto),
                        CONSTRAINT PRODUCTO_orden_FK FOREIGN KEY (id_producto) REFERENCES  PRODUCTO(id_producto) on delete
                        CASCADE,
                        CONSTRAINT Orden_orden_detalle_FK FOREIGN KEY (No_venta) REFERENCES  ORDEN_VENTA(No_venta)on delete CASCADE
                        );
```

Tabla EMAIL

```
CREATE TABLE EMAIL(
                    razon_cliente varchar(13) NOT NULL,
                    email varchar(100) NOT NULL,
                    CONSTRAINT EMAIL_PK PRIMARY KEY(razon_cliente,email),
                    CONSTRAINT CLIENTE_FK FOREIGN KEY (razon_cliente) REFERENCES CLIENTE (razon_cliente)
                    );
```

6.2 Funciones

Función nombre_si_da_codigo
Esta función devuelve el nombre de un producto dado su código de barras.

```
CREATE OR REPLACE FUNCTION nombre_si_da_codigo(codigo varchar(13))
RETURNS SETOF PRODUCTO AS $$
BEGIN
RETURN QUERY SELECT * FROM PRODUCTO where codigo_barras=codigo;
END
$$ LANGUAGE plpgsql;
```

Función reinicia2
Esta función reinicia el campo id_producto porque fue declarado como identity y si llega a haber un error se seguiría incrementando el campo.

```
CREATE OR REPLACE FUNCTION reinicia2()
RETURNS void AS $$
declare maxid int;
BEGIN
begin
select max(id_producto)+1 from PRODUCTO into maxid;
execute 'alter SEQUENCE producto_id_producto_seq  RESTART with '|| 1;
end;
END
$$ LANGUAGE plpgsql;
```

Función verifica_código_repetido
Esta función verifica que al momento de ingresar un producto en el inventario, éste no se encuentre registrado (producto repetido),

```
CREATE OR REPLACE FUNCTION verifica_codigo_repetido()
RETURNS trigger AS
$$
declare maxid int;
begin
if exists((SELECT codigo_barras,razon_proveedor FROM PRODUCTO group by codigo_barras,razon_proveedor having count(*)>1))then
raise notice 'tienes un producto con el mismo codigo y proveedor';
select (max(id_producto)-1) from PRODUCTO into maxid;
execute 'alter SEQUENCE producto_id_producto_seq  RESTART with '|| maxid;
delete from PRODUCTO where id_producto=(select (max(id_producto)) from PRODUCTO);
return new;
else
select max(id_producto)+1 from PRODUCTO into maxid;
execute 'alter SEQUENCE producto_id_producto_seq  RESTART with '|| maxid;
UPDATE PRODUCTO
set utilidad=(select precio_venta from PRODUCTO group by precio_venta having max(id_producto)=(select max(id_producto) from PRODUCTO))-
(select precio_compra from INVENTARIO
 where codigo_barras=(select codigo_barras from PRODUCTO group by codigo_barras having max(id_producto)=(select max(id_producto) from PRODUCTO)))
where id_producto=(select max(id_producto) from PRODUCTO);
raise notice 'se inserto correctamente';
return new;
end if;
END;
$$
LANGUAGE plpgsql;
```

Función verifica_orden_venta
Esta función concatena y da formato requerido en el proyecto al campo nota_venta.

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

Función verifica_codigo_repetido_before
Esta función sólo es para pruebas en caso de que se necesite verificar si hay un código (ID) repetido.

```
CREATE OR REPLACE FUNCTION verifica_codigo_repetido_before()
RETURNS trigger AS
$$
declare maxid int;
begin
if(exists(select id_producto from PRODUCTO)) then
return new;
else
return new;
end if;
END;
$$
LANGUAGE plpgsql;
```

Función verifica_borrado_producto
Cuando borramos la tabla "PRODUCTO" es necesario actualizar el "id_producto" ya que como el campo es identity, si teniamos datos y se borra la tabla entonces se queda almacenada la información.

```
CREATE OR REPLACE FUNCTION verifica_borrado_producto()
RETURNS trigger AS
$$
declare maxid int;
begin
if exists(select id_producto from PRODUCTO) then
select (max(id_producto)+1) from PRODUCTO into maxid;
execute 'alter SEQUENCE producto_id_producto_seq  RESTART with '|| maxid;
return new;
else
perform reinicia2();
return new;
end if;
END;
$$
LANGUAGE plpgsql;
```

Función verifica_borrado_orden_func
Esta función verifica la tabla ORDEN_VENTA y si se llega a borrar algo de la tabla se actuzaliza el ID de los productos.

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

Función verifica_borrado_orden_detalle_func
Verificamos la integridad de los datos de la tabla ORDEN_DETALLE, como su primary key es identity volvemos a actualizar su ID.

```
CREATE OR REPLACE FUNCTION verifica_borrado_orden_detalle_func()
RETURNS trigger AS
$$
declare maxid int;
begin
if(exists(select No_orden_detalle from ORDEN_DETALLE)) then
select (max(No_orden_detalle)+1) from ORDEN_DETALLE into maxid;
execute 'alter SEQUENCE orden_detalle_no_orden_detalle_seq  RESTART with '|| maxid;
return new;
else
execute 'alter SEQUENCE orden_detalle_no_orden_detalle_seq  RESTART with '|| 1;
return new;
end if;
END;
$$
LANGUAGE plpgsql;
```

Función verificar_orden
Esta función se encarga de varias cosas: detectar si no hay stock suficiente, actualizar el precio de un producto por si se ingresá mal, hace el cálculo de la cantidad a pagar por la venta, actualiza el inventario, y enviar un mensaje si hay un producto con menos de 3 unidades en stock.

```
CREATE OR REPLACE FUNCTION verificar_orden()
RETURNS trigger AS
$$
declare maxid int;
begin
if((select Unidades_stock -(select cantidad_articulo from ORDEN_DETALLE
                            where No_orden_detalle=(select No_orden_detalle from ORDEN_DETALLE  group by No_orden_detalle
                                                    having max(No_orden_detalle)=(select max(No_orden_detalle) from ORDEN_DETALLE )))
    from INVENTARIO where codigo_barras=(select codigo_barras from PRODUCTO P where id_producto=
                                        (select id_producto from ORDEN_DETALLE
                                         where No_orden_detalle=(select No_orden_detalle from ORDEN_DETALLE  group by No_orden_detalle
                                                                 having max(No_orden_detalle)=(select max(No_orden_detalle) from ORDEN_DETALLE )))))< 0) then
raise notice 'No se registrar la venta no hay inventario suficiente';
delete from ORDEN_DETALLE where No_orden_detalle=(select (max(No_orden_detalle)) from ORDEN_DETALLE);
return null;
else
UPDATE ORDEN_DETALLE
set precio_venta_producto=
(select precio_venta from PRODUCTO P where id_producto=
 (select id_producto
  from ORDEN_DETALLE
  where No_orden_detalle=(select No_orden_detalle from ORDEN_DETALLE  group by No_orden_detalle
                          having max(No_orden_detalle)=(select max(No_orden_detalle) from ORDEN_DETALLE ))))
where No_orden_detalle=(select max(No_orden_detalle) from ORDEN_DETALLE);

UPDATE ORDEN_DETALLE
set total_pagar=(select cantidad_articulo*(select precio_venta_producto from ORDEN_DETALLE
                                where No_orden_detalle=(select No_orden_detalle
                                      from ORDEN_DETALLE  group by No_orden_detalle having max(No_orden_detalle)=(select max(No_orden_detalle) from ORDEN_DETALLE )))
from ORDEN_DETALLE
where No_orden_detalle=(select max(No_orden_detalle) from ORDEN_DETALLE  group by No_orden_detalle
                        having max(No_orden_detalle)=(select max(No_orden_detalle) from ORDEN_DETALLE )))
where No_orden_detalle=(select max(No_orden_detalle) from ORDEN_DETALLE );

UPDATE INVENTARIO
set unidades_stock=
(select Unidades_stock -(select cantidad_articulo
                         from ORDEN_DETALLE
                         where No_orden_detalle=(select No_orden_detalle from ORDEN_DETALLE  group by No_orden_detalle
                                                 having max(No_orden_detalle)=(select max(No_orden_detalle) from ORDEN_DETALLE )))
 from INVENTARIO where codigo_barras=(select codigo_barras from PRODUCTO P where id_producto=
                                     (select id_producto
                                      from ORDEN_DETALLE
                                      where No_orden_detalle=(select No_orden_detalle from ORDEN_DETALLE  group by No_orden_detalle
                                                              having max(No_orden_detalle)=(select max(No_orden_detalle) from ORDEN_DETALLE )))))
where codigo_barras=(select codigo_barras from PRODUCTO P where id_producto=
                    (select id_producto
                     from ORDEN_DETALLE
                     where No_orden_detalle=(select No_orden_detalle from ORDEN_DETALLE  group by No_orden_detalle
                                             having max(No_orden_detalle)=(select max(No_orden_detalle) from ORDEN_DETALLE ))));

if((select Unidades_stock -(select cantidad_articulo
                            from ORDEN_DETALLE
                            where No_orden_detalle=(select No_orden_detalle from ORDEN_DETALLE  group by No_orden_detalle
                                                    having max(No_orden_detalle)=(select max(No_orden_detalle) from ORDEN_DETALLE )))
    from INVENTARIO where codigo_barras=(select codigo_barras from PRODUCTO P where id_producto=
                                        (select id_producto
                                         from ORDEN_DETALLE
                                         where No_orden_detalle=(select No_orden_detalle from ORDEN_DETALLE  group by No_orden_detalle
                                                                 having max(No_orden_detalle)=(select max(No_orden_detalle) from ORDEN_DETALLE )))))<=3) then
raise notice 'El stock de tu producto es menor a 3';
end if;
raise notice 'se inserto correctamente';
return new;
end if;
END;
$$
LANGUAGE plpgsql;
```

Función vista_información
Se crea una vista para cada orden de producto, dado que cada orden consta de diferente productos, en la tabla se muestran todos los productos existentes.

```
CREATE OR REPLACE FUNCTION vista_informacion(No_orden_cliente_recibida VARCHAR(8))
RETURNS  table(
                No_orden_cliente VARCHAR(8),
                fecha date,
                Nombre text,
                Producto varchar(40),
                Marca varchar(40),
                cantidad_articulo int,
                precio_cada_producto decimal(6,2),
                total_por_cada_prodcuto decimal(7,2)) AS $$
begin
return query (select ov.nota_venta,ov.fecha_venta,concat(c.nombre_pila,' ',c.apellido_p,' ',c.apellido_m),
              p.nombre,p.marca,od.cantidad_articulo,od.precio_venta_producto,od.total_pagar
              from ORDEN_VENTA as ov inner join
              ORDEN_DETALLE  as od on ov.No_venta=od.No_venta
              inner join CLIENTE as c on c.razon_cliente=ov.razon_cliente
              inner join producto as p on p.id_producto=od.id_producto
              where ov.nota_venta=No_orden_cliente_recibida);
END;
$$
LANGUAGE plpgsql;
```

Función vista_información_por_orden
Se crea otra vista pero ésta sólo contiene los artículos de la orden y el total a pagar sin ver todos los productos.

```
CREATE OR REPLACE FUNCTION vista_informacion_por_orden(No_orden_cliente_recibida varchar(8))
RETURNS  table(
              No_orden_cliente varchar(8),
              fecha date,
              Nombre text,
              cantidad_articulos bigint,
              total_pagar DECIMAL(7,2)) AS
$$
begin
return query (select ov.nota_venta,ov.fecha_venta,concat(c.nombre_pila,' ',c.apellido_p,' ',c.apellido_m),
             count(od.cantidad_articulo),SUM(od.total_pagar)
             from ORDEN_VENTA as ov inner join
             ORDEN_DETALLE  as od on ov.No_venta=od.No_venta
             inner join CLIENTE as c on c.razon_cliente=ov.razon_cliente
             where ov.nota_venta=No_orden_cliente_recibida group by ov.No_venta,ov.fecha_venta,c.nombre_pila,c.apellido_p,c.apellido_m);
END;
$$
LANGUAGE plpgsql;
```

Función stock_menor_3
Esta función nos devuelve los productos que tengan un stock menor a 3 en el inventario

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

```

Función cantidad_vendida_por_fecha
Esta función toma como parámetros una fecha de inicio y otra fecha posterior a ésta, y nos devuelve el total vendido en este periodo.

```
CREATE OR REPLACE FUNCTION CANTIDAD_VENDIDA_POR_FECHA(FECHA_INICIO DATE, FECHA_FIN DATE)
RETURNS  table(cantidad_vendida  numeric)
as
$$
begin
return query (select sum(od.total_pagar)
              from ORDEN_VENTA as ov inner join
              ORDEN_DETALLE  as od on ov.No_venta=od.No_venta
              where ov.No_venta=od.No_venta and ov.fecha_venta between FECHA_INICIO and FECHA_FIN);
END;
$$
LANGUAGE plpgsql;
```

6.3 Triggers

Trigger after_trigger
Este trigger sirve para ejecutar la función verifica_codigo_repetido() y cerciorarse cuándo se haga el insert.

```
CREATE TRIGGER after_trigger
after insert
ON PRODUCTO
FOR EACH ROW
EXECUTE PROCEDURE verifica_codigo_repetido();
```

Trigger trigger_borrado_producto_actualiza_id
Este trigger es para si se llaga hacer el 'delete from producto' y llame a la función 'verifica_borrado_producto'

```
CREATE TRIGGER trigger_borrado_producto_actualiza_id
after delete
ON PRODUCTO
FOR EACH ROW
EXECUTE PROCEDURE verifica_borrado_producto();
```

Trigger trigger_borrado_orden
Este trigger corresponde a la función verifica_borrado_orden_func() por si se llega a hacer algún borrado.

```
1  CREATE TRIGGER trigger_borrado_orden
2  after delete
3  ON ORDEN_VENTA
4  FOR EACH ROW
5  EXECUTE PROCEDURE verifica_borrado_orden_func();
6
```

Trigger trigger_borrado_orden_detalle
Este trigger corresponde a la función verifica_borrado_orden_detalle_func por si se llega a borrar la tabla ORDEN_DETALLE.

```
1  CREATE TRIGGER trigger_borrado_orden_detalle
2  after delete
3  ON ORDEN_DETALLE
4  FOR EACH ROW
5  EXECUTE PROCEDURE verifica_borrado_orden_detalle_func();
```

Trigger before_trigger_orden
Este trigger se crea antes de que se haga la inserción de la tabla "ORDEN" para que le de formato antes de insertar.

```
1  CREATE TRIGGER before_trigger_orden
2  before insert
3  ON ORDEN_VENTA
4  FOR EACH ROW
5  EXECUTE PROCEDURE verifica_orden_venta();
```

Trigger before_trigger
El trigger se crea antes de la función "insert_before()" y se llama a la función para que haga su correspondiente proceso.

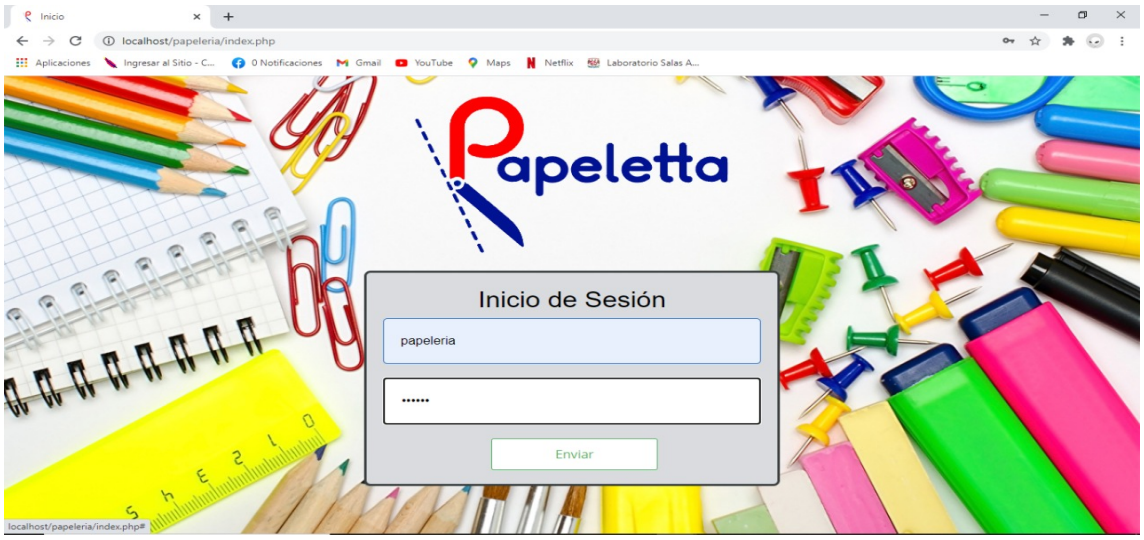
```
1  CREATE TRIGGER before_trigger
2  before insert
3  ON PRODUCTO
4  FOR EACH ROW
5  EXECUTE PROCEDURE verifica_codigo_repetido_before();
6
```

Trigger test_trigger
Este trigger corresponde al insert en orden_detalle, que es la venta del producto.

```
1  CREATE TRIGGER test_trigger
2  AFTER INSERT
3  ON ORDEN_DETALLE
4  FOR EACH ROW
5  EXECUTE PROCEDURE verificar_orden();
```

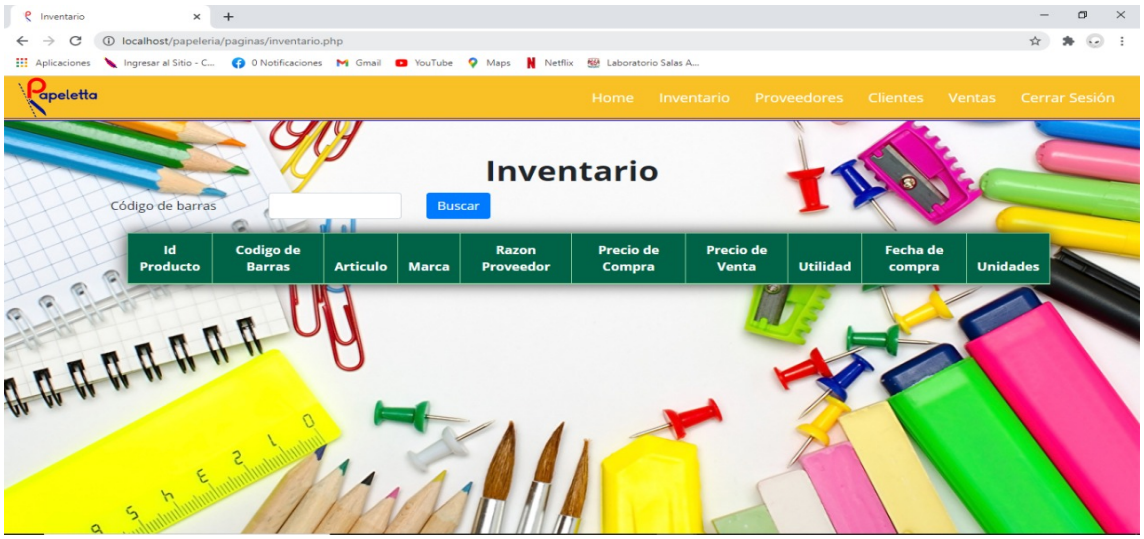
7 Implementación de Página Web

Inicio de sesión de la página web

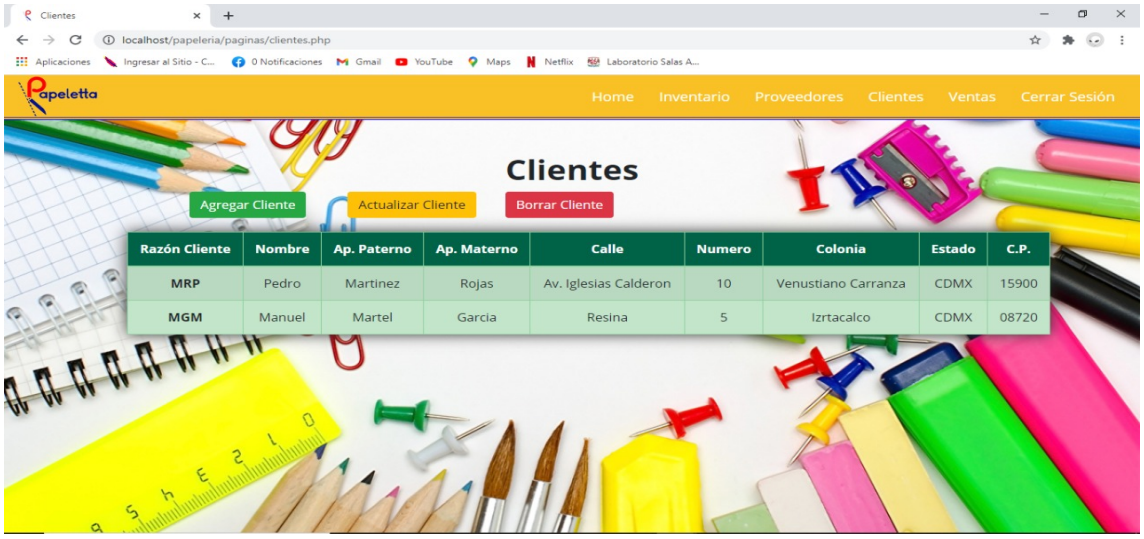


Página principal

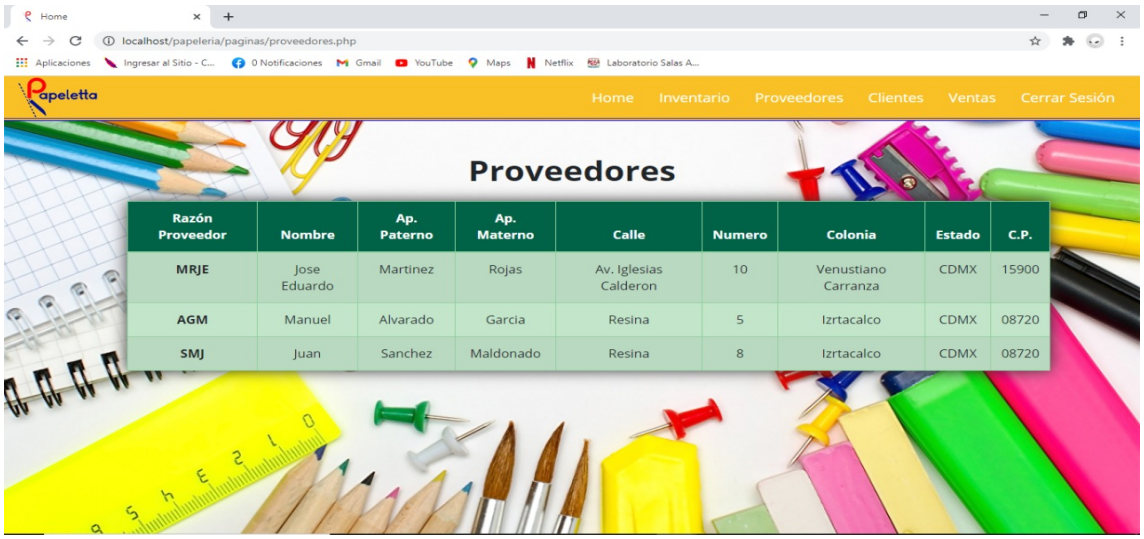
Inventario



Clientes



Proveedores



8 Conclusiones

8.1 Martínez Rojas José Eduardo

En resumidas cuentas se pudo resolver el problema, aunque con algunas dificultades en cuanto investigación de temas, pero que finalmente se pudieron resolver. En cuanto al diseño de este proyecto me pareció una muy buena manera de aplicar todos los conocimientos vistos en clase y laboratorio, y poderlos hacer los prácticos a la realidad. Me pareció muy agradable este proyecto debido a que ya había trabajado en un proyecto pequeño para ventas familiar, aunque sin el uso de base de datos que le da mejor diseño y solución a muchos problemas que no había detectado con anterioridad. Finalmente puedo decir que me gustó trabajar en equipo con mis compañeros y que a pesar de las condiciones en línea se pudo llevar con éxito este proyecto.

8.2 Vázquez Torres Juan Adrián

El proyecto me pareció interesante debido a que tanto lo que se pedía, como el formato y los requisitos a entreger fueron algo atípicos, al menos hablando por mí. Fue una manera eficaz de poner a prueba todos los conocimientos que adquirimos durante el semestre tanto en la clase de teoría como en el laboratorio. Además el profesor nos pidió que expusieramos el proyecta como si se los estuviéramos vendiendo a un cliente, por lo que creo que este proyecto fue una buena manera de prepararnos para escenarios que pudiéramos enfrentar en un futuro en nuestra vida laboral.

8.3 Báez Cadena Diestefano Michel

Este proyecto ha sido una gran experiencia, ya que va más allá de una simple bases de datos, requirió de la creación de una empresa, una página web y la realización de la carga de la base de datos a la página, para después "venderla" a un cliente; todo esto nos da idea de como es que se trabaja en la vida real y por todos los pasos que conlleva un proyecto, desde el software, hasta la documentación. En lo personal lo más complicado fue la carga de la base de datos a la página puesto que ya tengo conocimientos básicos de creaciones de páginas, pero nunca había cargado una base a las páginas. Me llevo un gran aprendizaje en muchos aspectos, así como el gusto de haber trabajado con mi equipo.