Universidad Nacional Autónoma de México

Facultad de Ingeniería

Bases de Datos

Proyecto final

Semestre 2022-1

Fecha entrega: 11/12/2021

Castro Alonso Jéssica
Garces Gomez Eduardo Tonathiu
Martín Alejandro Ávalos Medina
Osorio Alvarado Jorge Adalberto

Grupo 01

Profesor: Ing. Fernando Arreola Franco

Indice de secciones

1. Objetivo proyecto

El alumno analizará una serie de requerimientos y propondrá una solución que atienda a los mismos, aplicando los conceptos vistos en el curso.

2. Introducción

A lo largo del curso se aprendió a diseñar e implementar sistemas de bases de datos relacionales. El objetivo de este proyecto consiste en aplicar todos los conceptos y principios vistos en clase para solucionar un problema, el cual, consiste en el diseño de una base de datos para una cadena de papelerías. Para el análisis del problema primero se identificaron los atributos, entidades y relaciones para construir el modelo conceptual. Se identificaron entidades para almacenar los datos del proveedor, almacenar lo datos del producto que suministra el proveedor, guadar la categoría a la que pertenece el cada producto, almacenar los datos de cada artículo por cada producto, guardar los datos correspondientes a una venta y, por último, almacenar los datos del cliente. Se propone tener tres atributos derivados: pago final de la venta, pago por cada artículo y la cantidad total de artículos, para lo cuales, se utilizaron algunos cálculos con procedimientos y disparadores en la implementación del modelo físico. Una vez identificados los tipos de atributos y tipos de relaciones, se realizó el modelo lógico, donde se definieron las tablas a utilizar para el modelo físico. Se propone, para las tablas, llaves primarias artificiales con el fin de que, por medio de un secuenciador, se generen de manera automática al momento de realizar inserciones de datos. Por ultimo, por medio del lenguaje de definición de datos, se crearon las tablas con sus respectivas restricciones de integridad y se utilizó el lenguaje de manipulación de datos para realizar inserciones a las tablas para verificar el correcto funcionamiento de la base de datos.

3. Plan de trabajo

3.1. Información del Proyecto

El proyecto sera realizado por alumnos del la Universidad Nacional Autónoma de México de la Facultad de Ingeniería en la carrera de Ingeniería en Computación el proyecto es nombrado como BD Papelería.
El plan de trabajo fue elaborado el día 28 de noviembre del año 2021 y el periodo que abarco fue de 2 semanas hasta la entrega del mismo.

3.2. Propósito y justificación del proyecto

La elaboración del proyecto es con el fin de entregar un producto a nuestro cliente en este caso la elaboración de una base de datos para que nuestro cliente pueda tener un control con respecto a su negocio esto dio origen a la necesidad de poder saber quienes son los proveedores que le proporcionan producto que cantidad compro y a que precio, también saber que productos son los que fueron vendidos cuantos le quedan para así poder hacer pedidos en algún futuro, no conforme con esto saber que clientes les compran y de donde pertenecen, suponiendo que en algún momento lleguen a enviar pedidos a domicilio no necesiten mas información de la que ya a sido proporcionada al haberse registrado.

3.3. Resumen del proyecto

3.3.1. Objetivo

Para lograr el propósito de nuestro proyecto primero debemos encontrar el modelo que utilizaremos y tomarlo de base en este caso nos concentramos en utilizar el modelo entidad relación por lo mismo necesitamos entender que información queremos guardar y de quien en especifico la queremos guardar, segundo una vez entendido esto y habiendo elaborado nuestro modelo entidad-relación crearemos la elaboración de nuestra base de datos conceptual con el modelo relacional, tercero utilizar el modelo relacional para la creación de la base de datos física por ultimo probarla para ver el comportamiento de la misma.
Los medios para lograr lo anterior es utilizar software que nos ayude a la creación de cada modelo que se llegara a obtener por lo que se usara el software Día para elaborar nuestro modelo entidad-relación, para el modelo relacional ERStudio por ultimo para crear la base de datos física se utilizara el manejador postgreSQL, también preguntaremos al cliente en especifico lo que requiere desea para su completa implementacion teniendo la posibilidad de que se propongan algunos atributos o entidades que no se hayan propuesto por el mismo.

3.3.2. Actividades a realizar

Para las actividades que se planean tener y la fecha en la que se tiene pensado iniciar y acabar son las siguientes:
Del día 28 al 29 de noviembre identificar los requerimientos del cliente por lo que se preguntara la información que quiera guardar y de quien, necesitamos saber si es que es obligatorio guardar todos los datos o si bien pueden ser opcionales.
Del 30 al 4 de diciembre se elaborara el modelo entidad-relación mediante el uso de el software Día para saber que acabamos la actividad es observando gráficamente los tipos de datos, las entidades a las que pertenece y como se relacionan. Del 5 al 6 de diciembre se elaborara el modelo relacional utilizando la herramienta ERStudio por lo que la actividad terminara una ves que hayamos transformado nuestro modelo entidad-relación a modelo relacional por lo que sabremos cuales son las nuevas relaciones o tablas, como estarán conformadas, si se legaron a proponer nuevos atributos y las restricciones de cada uno.
Del 7 al 8 de diciembre se creara la base de datos en el manejador PostgreSQL por lo que podemos decir que se construyo físicamente basándonos en el modelo relacional usando el lenguaje para el mismo la forma en la que se habrá terminado esta sección es habiendo especificado las funciones y consultas que necesitamos así como las tablas que deben conformar el mismo.

3.3.3. Recursos

Humanos
Se necesitaran 4 integrantes para la realización del proyecto este personal deberán ser ingenieros en computación la forma en la que realizaran cada actividad es con la completa participación del equipo.
Infraestructura
Las instalaciones que se ocuparon fueron cada uno en su casa y la forma en la que hubo comunicación fue mediante el teléfono celular.
Material y equipos

Lo que se utilizo fue como ya se había especificado anteriormente software como Día ERStudio y PostgreSQL y por ultimo también se utilizaron computadoras para la programación.

3.3.4. Presupuesto y financiamiento

No hubo un presupuesto especifico sigue en espera de confirmar.

3.3.5. Evaluación

El comité que se encarga de aceptar este plan de trabajo esta analizando el documento y estamos en espera de saber si lo aceptaran o pedirán mas información.

3.3.6. Firmas

Aquí serán entregadas las firmas que de los miembros del comité.

4. Diseño

4.1. Descripción del problema

Una cadena de papelerías busca innovar la manera en que almacena su información. Se requieren de sistemas informáticos para almacenar datos como la razón social, domicilio, nombre y teléfonos de los proveedores, rfc, nombre, domicilio y al menos un email de los clientes. También se requiere 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.

4.2. Requerimientos y consideraciones

- Almacenar razón social, domicilio, nombre y teléfono de los proveedores.
- RFC, nombre, domicilio y al menos un email de los clientes.
- Guardar 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.
- También se guardará el numero de venta, fecha de venta y precio total a pagar por cada artículo.
- Al recibir el código de barras de un producto, regresar la utilidad.
- Al venderse un articulo, se deberá decrementar el stock por la cantidad vendida de ese artículo. Si el valor llega a cero, abortar la transacción. Si el pedido se completa pero quedan menos de 3 en stock, se emitirá una alerta.
- Al recibir una fecha, o una fecha de inicio y una fecha final, se regresará la cantidad total que se vendió y la ganancia correspondiente en esa fecha/periodo.
- Permitir obtener el nombre de aquellos productos con menos de 3 ejemplares en el stock.
- Deberá generarse de forma automática una vista que contenga la 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 formato de la venta deberá ser como se muestra a continuación "VENT-001".
- En donde se presente el atributo domicilio, deberá estar compuesto por estado, código postal, colonia, calle y número.
- El diseño deberá satisfacer todos los principios de diseño los requerimientos anteriores y un buen manejo de la información.

4.3. Identificación de entidades y atributos

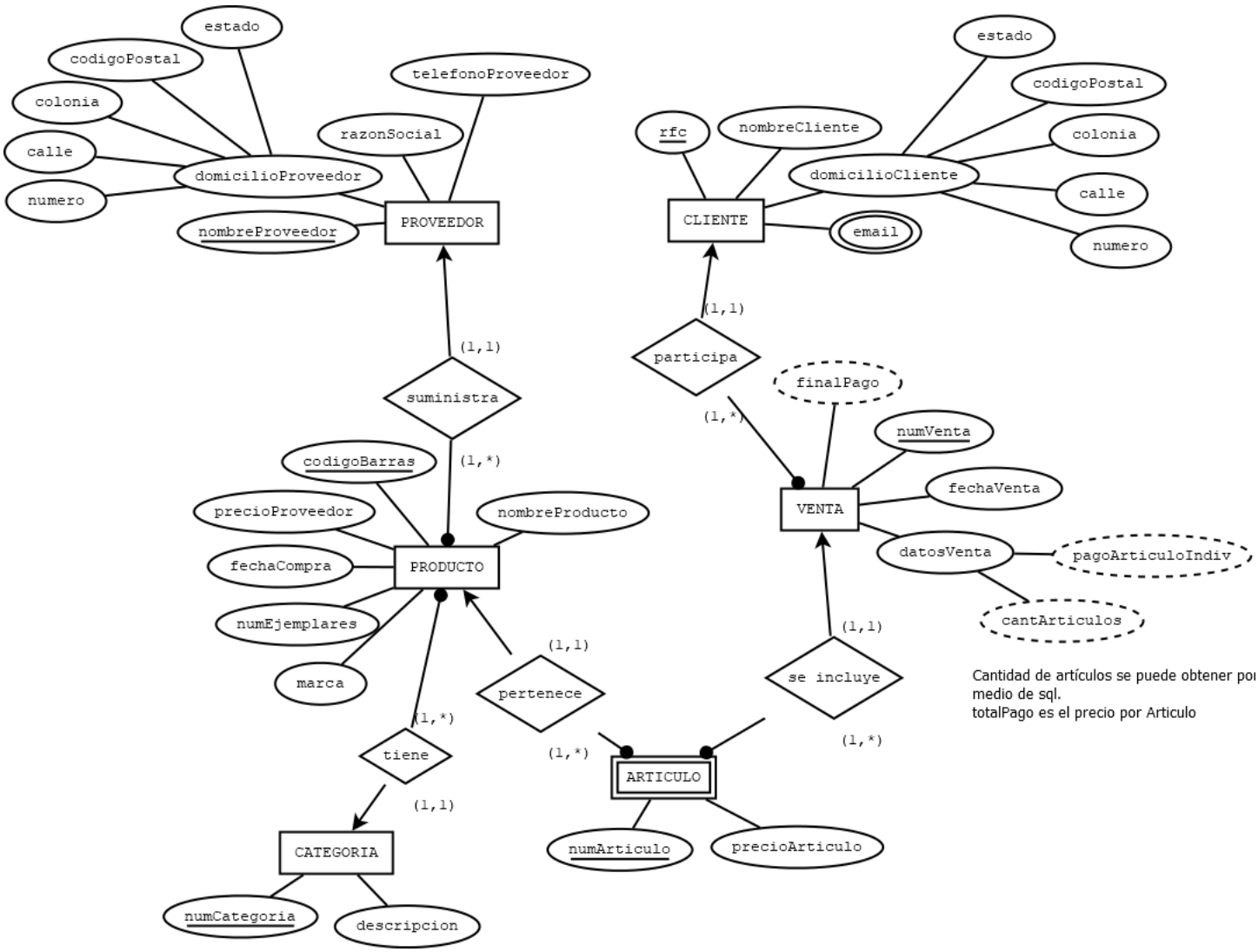
Para identificar a las entidades, se reconocieron a los objetos y abstracciones de los cuáles se requería guardar información. Los atributos se identificaron relacionándolos como características de las entidades:

- **Proveedor** (nombre del **proveedor**, domicilio del **proveedor**, razon social del **proveedor** y teléfono del **proveedor**).
- Cliente (RFC del **cliente**, nombre del **cliente**, domicilio del **cliente** y email del **cliente**).
- Articulo (número de **artículo** y precio por **artículo**).
- Producto (código de barras del **producto**, nombre del **producto**, precio del **producto** al que fue comprado al proveedor (por pieza), fecha de la compra del **producto**, número de ejemplares en stock del **producto** y marca del **producto**).
- Venta (número de **venta**, fecha de la **venta**, pago final de la **venta** y cantidad de artículos en la **venta**).
- Categoría (número de **categoría** y descripción de la **categoría**).

Al identificar a la entidad **Artículo**, notamos que su clave principal no era suficiente para distinguirla, por lo que determinamos que sería una entidad débil. Para solucionar esto, se propuso que por cada clave principal de la entidad **Producto**, se reiniciara la clave principal de la entidad débil **Artículo**. Por otra parte, también se identificaron atributos cuyos valores cambiaban conforme se insertaban ciertos valores. Éstos atributos pertenecen a la entidad **Venta** y se planeó solucionarlo por medio de disparadores (triggers) y procedimientos.

4.4. Modelo conceptual

Basándonos en la descripción del problema y teniendo en cuenta las entidades y atributos determinados anteriormente, identificamos las relaciones entre las entidades. Con base en esto, creamos el modelo conceptual (para el cual hicimos uso del software Dia, que es una herramienta útil para crear nuestro modelo entidad-relación):



Cada entidad tiene sus correspondientes atributos y definimos cada uno de ellos con sus claves principales. También definimos al atributo email como multivaluado porque existe la opción de que el cliente agregue más de un email, pero es atributo obligatorio porque debe haber registrado por lo menos uno.

Para las cardinalidades tenemos que para el proveedor y el producto se relacionan con suministra y la forma en la que se puede leer es de la siguiente forma. Un proveedor suministra uno o muchos productos pero un producto es suministrado por un proveedor, teniendo así una relación de uno a muchos.

Para producto con categoría se leería como un producto tiene una categoría y una categoría tiene muchos productos, teniendo una cardinalidad de muchos a uno.

Producto también esta relacionado con artículo, para éste caso, la relación se leería como: un producto pertenece a muchos artículos pero un articulo pertenece a un producto, teniendo una relación de uno a muchos.
Para la entidad venta y articulo, se relacionan de la siguiente forma: una venta tiene muchos artículos pero un articulo pertenece a una venta, la cardinalidad es de uno a muchos.
Por último, la relación cliente y venta es: un cliente participa en muchas ventas y en una venta participa un cliente, con una cardinalidad de uno a muchos.

4.5. Modelo lógico

El modelo Relacional se implemento teniendo en cuenta las siguientes consideraciones:

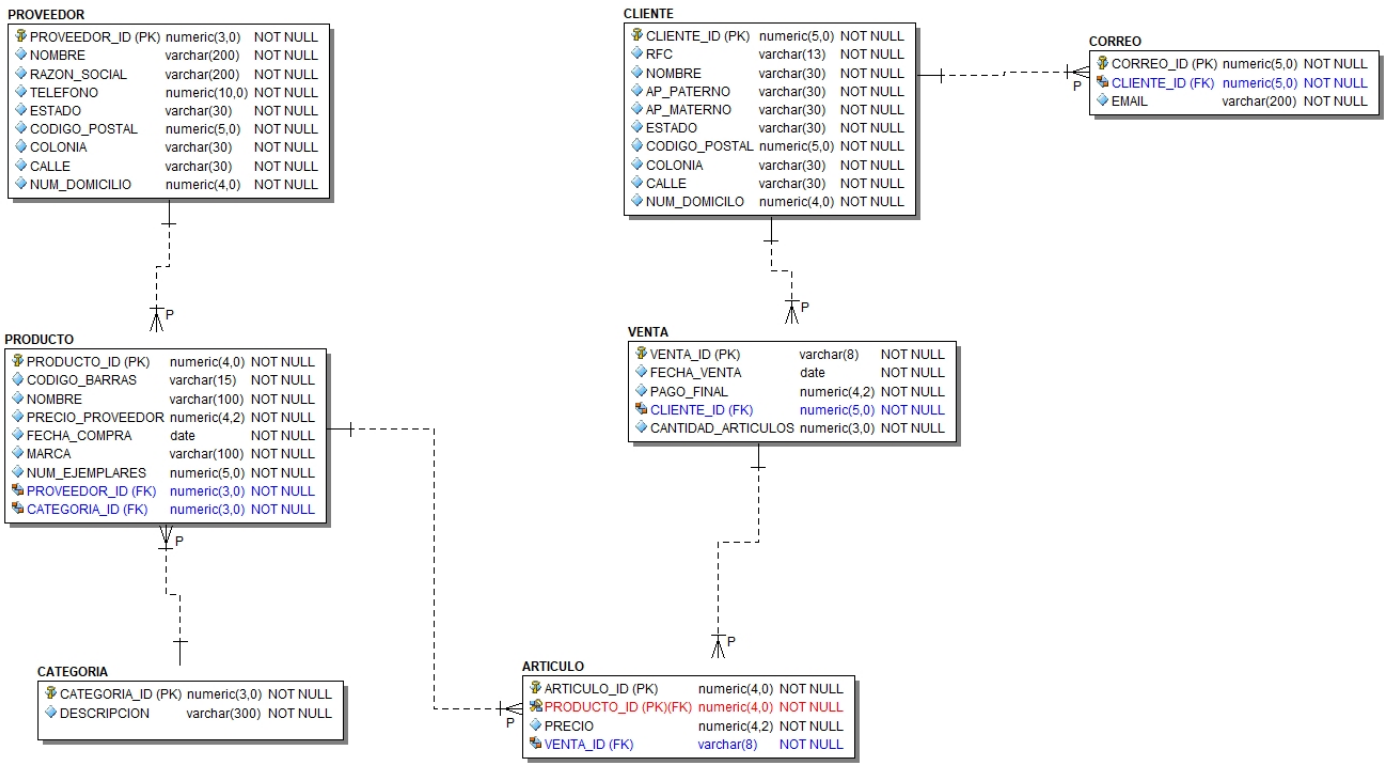
• Para que exista un producto, debe de existir al menos un proveedor, un producto es solamente distribuido por un proveedor, pero un proveedor puede vender varios productos.

• Si existen productos, existen categorías, en donde un producto pertenece a una categoría y en una categoría se pueden tener uno o más productos.

• De los productos derivan los artículos, de un mismo producto puede haber una o más existencias llamadas artículos.

• Una venta debe contener uno o mas artículos, y a cada venta le corresponde un cliente asociado a ella, pero un cliente puede registrar varias ventas.

• Un cliente puede tener uno o más correos electrónicos.



5. Implementación

5.1. DDL

```
CREATE TABLE PROVEEDOR(
    PROVEEDOR_ID   smallint(3,0)     NOT NULL,
    NOMBRE         varchar(200)      NOT NULL,
    RAZON_SOCIAL   varchar(200)      NOT NULL,
    TELEFONO       numeric(10,0)     NOT NULL,
    ESTADO         varchar(30)       NOT NULL,
    CODIGO_POSTAL  smallint(5,0)      NOT NULL,
    COLONIA        varchar(30)       NOT NULL,
    CALLE          varchar(30)       NOT NULL,
    NUM_DOMICILIO  smallint(5,0)      NOT NULL,
    CONSTRAINT PROVEEDOR_PK PRIMARY KEY (PROVEEDOR_ID)
);

CREATE TABLE CLIENTE(
    CLIENTE_ID     smallint(5,0)    NOT NULL,
    RFC            varchar(13)      NOT NULL,
    NOMBRE         varchar(30)      NOT NULL,
    AP_PATERNO     varchar(30)      NOT NULL,
    AP_MATERNO     varchar(30),
    ESTADO         varchar(30)      NOT NULL,
    CODIGO_POSTAL  smallint(5,0)    NOT NULL,
    COLONIA        varchar(30)      NOT NULL,
    CALLE          varchar(30)      NOT NULL,
    NUM_DOMICILO   smallint(5,0)    NOT NULL,
    CONSTRAINT CLIENTE_PK PRIMARY KEY (CLIENTE_ID)
);

CREATE TABLE CATEGORIA(
    CATEGORIA_ID  smallint(3,0)    NOT NULL,
    DESCRIPCION   varchar(300)     NOT NULL,
    CONSTRAINT CATEGORIA_PK PRIMARY KEY (CATEGORIA_ID)
);

CREATE TABLE CORREO(
    CORREO_ID   numeric(5,0)     NOT NULL,
    CLIENTE_ID  numeric(5,0)     NOT NULL,
    EMAIL       varchar(200)     NOT NULL,
    CONSTRAINT CORREO_PK PRIMARY KEY (CORREO_ID),
CONSTRAINT CORREO_CLIENTE_ID_FK FOREIGN KEY (CLIENTE_ID)
REFERENCES CLIENTE (CLIENTE_ID),
CONSTRAINT CORREO_CLIENTE_ID_EMAIL_UK UNIQUE (CLIENTE_ID,EMAIL)
);
```

```
CREATE TABLE VENTA(
    VENTA_ID           varchar(8)       NOT NULL,
    FECHA_VENTA        date             NOT NULL,
    PAGO_FINAL         numeric(6,2)     NOT NULL,
    CLIENTE_ID         numeric(5,0)     NOT NULL,
CANTIDAD_ARTICULOS smallint (3,0)   NOT NULL,
    CONSTRAINT VENTA_PK PRIMARY KEY (VENTA_ID),
CONSTRAINT VENTA_CLIENTE_ID_FK FOREIGN KEY (CLIENTE_ID)
REFERENCES CLIENTE (CLIENTE_ID)
);

CREATE TABLE PRODUCTO(
    PRODUCTO_ID       smallint(4,0)    NOT NULL,
    CODIGO_BARRAS     varchar(15)      NOT NULL,
    NOMBRE            varchar(100)     NOT NULL,
    PRECIO_PROVEEDOR  numeric(4,2)     NOT NULL,--por pieza
    FECHA_COMPRA      date             NOT NULL,
    MARCA             varchar(100)     NOT NULL,
    NUM_EJEMPLARES    numeric(5,0)     NOT NULL,
    PROVEEDOR_ID      smallint(3,0)    NOT NULL,
    CATEGORIA_ID      smallint(3,0)    NOT NULL,
    CONSTRAINT PRODUCTO_PK PRIMARY KEY (PRODUCTO_ID),
CONSTRAINT PRODUCTO_PROVEEDOR_ID_FK FOREIGN KEY (PROVEEDOR_ID)
REFERENCES PROVEEDOR (PROVEEDOR_ID),
CONSTRAINT PRODUCTO_CATEGORIA_ID_FK FOREIGN KEY (CATEGORIA_ID)
REFERENCES CATEGORIA (CATEGORIA_ID)
);

CREATE TABLE ARTICULO(
    ARTICULO_ID  smallint(4,0)   NOT NULL,
    PRODUCTO_ID  smallint(4,0)   NOT NULL,
    PRECIO       numeric(4,2)    NOT NULL,
    VENTA_ID     varchar(8)      NOT NULL,
    CONSTRAINT ARTICULO_PK PRIMARY KEY (ARTICULO_ID, PRODUCTO_ID),
CONSTRAINT ARTICULO_PRODUCTO_ID_FK FOREIGN KEY (PRODUCTO_ID)
REFERENCES PRODUCTO (PRODUCTO_ID),
CONSTRAINT ARTICULO_VENTA_ID_FK FOREIGN KEY (VENTA_ID)
REFERENCES VENTA (VENTA_ID)
);

CREATE SEQUENCE PROVEEDOR_PROVEEDOR_ID_SEQ
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
```

```
ALTER TABLE PROVEEDOR ALTER COLUMN PROVEEDOR_ID SET DEFAULT
 nextval('PROVEEDOR_PROVEEDOR_ID_SEQ'::regclass);

CREATE SEQUENCE CLIENTE_CLIENTE_ID_SEQ
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE CLIENTE ALTER COLUMN CLIENTE_ID SET DEFAULT
 nextval('CLIENTE_CLIENTE_ID_SEQ'::regclass);

CREATE SEQUENCE CATEGORIA_CATEGORIA_ID_SEQ
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE CATEGORIA ALTER COLUMN CATEGORIA_ID SET DEFAULT
 nextval('CATEGORIA_CATEGORIA_ID_SEQ'::regclass);

CREATE SEQUENCE CORREO_CORREO_ID_SEQ
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE CORREO ALTER COLUMN CORREO_ID SET DEFAULT
 nextval('CORREO_CORREO_ID_SEQ'::regclass);

CREATE SEQUENCE PRODUCTO_PRODUCTO_ID_SEQ
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE PRODUCTO ALTER COLUMN PRODUCTO_ID SET DEFAULT
 nextval('PRODUCTO_PRODUCTO_ID_SEQ'::regclass);

alter table venta add constraint ck_venta_venta_id
 check(venta_id like ('VENT-%'));
```

```
alter table venta alter column pago_final set default '0.0';
alter table venta alter column cantidad_articulos set default '0';
```

5.2. Vistas

Vista para factura:

```
create view vst_factura as
select
  cl.nombre as nombre_cliente,cl.ap_paterno,
  co.email,a.venta_id,v.fecha_venta,p.nombre as nombre_articulo,
  p.marca,a.precio,count(*) as cantidad,v.cantidad_articulos,v.pago_final
from articulo a join producto p on a.producto_id=p.producto_id
                join venta v on a.venta_id=v.venta_id
                join cliente cl on v.cliente_id=cl.cliente_id
                join correo co on cl.cliente_id=co.cliente_id
group by a.venta_id,v.fecha_venta,cl.nombre,cl.ap_paterno,
         co.email,v.cantidad_articulos,v.pago_final,
         p.nombre,p.marca,a.precio
order by a.venta_id;
```

Ejemplo de uso:

```
select * from vst_factura where venta_id='VENT-011';
```

Vista para poder saber los valores a insertar en un nuevo articulo:

```
create view vst_informacion_para_registrar_articulo as
select
  p.producto_id,p.nombre,p.marca,p.precio_proveedor,
  a.precio as precio_vendido,max(a.articulo_id) as articulo_id_max
from articulo a right join producto p on a.producto_id=p.producto_id
group by p.producto_id,p.nombre,p.precio_proveedor,a.precio
order by p.producto_id;
```

Ejemplo de uso:

```
select * from vst_informacion_para_registrar_articulo;
```

Vista para obtener el nombre de los productos con pocos ejemplares:

```
create view vst_productos_escasos as
select nombre,marca from producto where num_ejemplares<'3';
```

Ejemplo de uso:

```
select * from vst_productos_escasos;
```

5.3. Índice

Como la consulta para saber la utilidad de un producto, dado su código de barras, es solicitada y suponemos que con mucha frecuencia, con la finalidad de reducir el número de lecturas para recuperar los datos producto_id y precio_proveedor, se crea un índice de tipo unique y con el método BTREE para la columna codigo_barras de la tabla producto. Se especifica que se realice el índice con el método BTREE porque éste tiene sus datos ordenados, no permite datos nulos y su crecimiento evita que el árbol tenga un alto grado de profundidad (con lo cual se reduce el número de lecturas):

```
create unique index producto_codigo_barras_iuk on producto
 using btree(codigo_barras);
```

5.4. Procedimientos y disparadores

Procedimiento: agregar_articulo_a_venta(prod smallint,vent varchar(8))
Parámetros: producto_id del producto que se quiere agregar, venta_id de la venta a la cual se va a agregar el artículo. Se le aplica una utilidad del 30 % por default. También se decrementa el número de ejemplares en la tabla producto, si hay 0 ejemplares del producto añadido, se aborta la transacción.

```
CREATE OR REPLACE PROCEDURE
 agregar_articulo_a_venta(prod smallint,vent varchar(8))
LANGUAGE PLPGSQL
AS $$
  DECLARE
    art smallint;
    preC numeric(6,2);
    preI numeric(6,2);
    num_ejem integer;
  BEGIN
    IF (select articulo_id_max from vst_informacion_para_registrar_articulo
        where producto_id=prod) IS NOT NULL THEN
          art=(select articulo_id_max from
           vst_informacion_para_registrar_articulo
        where producto_id=prod)+1;
          preI=(select precio_vendido from
           vst_informacion_para_registrar_articulo
        where producto_id=prod);
    ELSE
        art=1;
        preC=(select precio_proveedor from producto where producto_id=prod);
        preI=preC+((30*preC)/100);
    END IF;
    insert into articulo (articulo_id,producto_id,precio,venta_id)
      values (art,prod,preI,vent);
    num_ejem=(select num_ejemplares from producto where producto_id=prod);
    update producto set num_ejemplares=(num_ejem-1) where producto_id=prod;
    IF num_ejem > 0 THEN
      IF num_ejem <= 4 THEN
```

```
        RAISE NOTICE
        'ALERTA, QUEDAN % ARTICULOS DEL PRODUCTO SOLICITADO',num_ejem-1;
      END IF;
      commit;
    ELSE
      RAISE NOTICE 'No existen ejemplares del producto % por el momento.',prod;
      rollback;
    END IF;
  END;
$$;
```

Procedimiento: calcula_cant_articulos_total_por_venta().

```
CREATE OR REPLACE FUNCTION calcula_cant_articulos_total_por_venta()
RETURNS TRIGGER
AS $$
  DECLARE
    v_venta_id venta.venta_id%type;
    v_cant venta.CANTIDAD_ARTICULOS%type;
  BEGIN
    v_venta_id := NEW.venta_id;
    RAISE NOTICE '%',v_venta_id;
    v_cant=(SELECT count(*) as cantidad_articulos_total
    FROM venta v join articulo a on v.venta_id=a.venta_id
    GROUP BY V.venta_id,v.fecha_venta HAVING v.venta_id=v_venta_id);

    update venta set CANTIDAD_ARTICULOS=v_cant where venta.venta_id=v_venta_id;
  RETURN NEW;
  END;
$$ LANGUAGE PLPGSQL;
```

Trigger: tr_calcula_cant_articulos_total_por_venta().

```
CREATE TRIGGER tr_calcula_cant_articulos_total_por_venta
  AFTER INSERT ON ARTICULO FOR EACH ROW
  EXECUTE PROCEDURE calcula_cant_articulos_total_por_venta();
```

Procedimiento: calcula_pago_total_por_venta().

```
CREATE OR REPLACE FUNCTION calcula_pago_total_por_venta()
RETURNS TRIGGER
AS $$
  DECLARE
    v_venta_id venta.venta_id%type;
    v_producto_id articulo.producto_id%type;
    v_pago venta.pago_final%type;
--    v_cant_indv venta.cantidad_articulos%type;
  BEGIN
    v_venta_id := NEW.venta_id;
```

```
    v_producto_id := NEW.producto_id;
    v_pago=(select a.precio
      from articulo a join producto p on a.producto_id=p.producto_id
      group by a.venta_id,a.producto_id,p.nombre,p.marca,a.precio
      HAVING a.venta_id=v_venta_id and a.producto_id=v_producto_id);
--    v_cant_indv=(select count(*) as cantidad
--      from articulo a join producto p on a.producto_id=p.producto_id
--      group by a.venta_id,a.producto_id,p.nombre,p.marca,a.precio
--      HAVING a.venta_id=v_venta_id and a.producto_id=v_producto_id);

      update venta set PAGO_FINAL=venta.pago_final+v_pago
       where venta.venta_id=v_venta_id;
  RETURN NEW;
  END;
$$ LANGUAGE PLPGSQL;
```

Trigger: tr_calcula_pago_total_por_venta().

```
CREATE TRIGGER tr_calcula_pago_total_por_venta
  AFTER INSERT ON ARTICULO FOR EACH ROW
  EXECUTE PROCEDURE calcula_pago_total_por_venta();
```

Procedimiento: utilidad(cod_barras varchar(15))

```
CREATE OR REPLACE PROCEDURE utilidad(cod_barras varchar(15))
LANGUAGE PLPGSQL
AS $$
  DECLARE
    num_ejem integer;
    v_producto_id producto.producto_id%type;
    v_precio_prov producto.precio_proveedor%type;
    v_precio_vend articulo.precio%type;
    v_utilidad numeric(6,2);
  BEGIN
    v_producto_id=(select producto_id from producto
     where codigo_barras=cod_barras);
    v_precio_prov=(select precio_proveedor from producto
     where codigo_barras=cod_barras);
    v_precio_vend=(select max(precio) from articulo
     group by producto_id having producto_id=v_producto_id);
    v_utilidad=((v_precio_vend-v_precio_prov)*100)/v_precio_prov;

    RAISE NOTICE 'Producto: %',v_producto_id;
    RAISE NOTICE 'Precio proveedor: % pesos',v_precio_prov;
    RAISE NOTICE 'Precio vendido: % pesos',v_precio_vend;
    RAISE NOTICE 'Utilidad: % por ciento',v_utilidad;
  END;
$$;
```

Ejemplo de llamada al procedimiento:

```
CALL utilidad('A-00050-Z');
```

Procedimiento: venta_durante_temporada(fec_i date,fec_f date) para obtener el total articulos vendidos entre las fechas 'YYYY-MM-DD' y 'YYYY-MM-DD'.

```
CREATE OR REPLACE PROCEDURE venta_durante_temporada(fec_i date,fec_f date)
LANGUAGE PLPGSQL
AS $$
  DECLARE
    cur_datos_cant_articulos CURSOR FOR
      select fecha_venta,sum(cantidad_articulos) as cant_total_art
      from venta group by fecha_venta
      having fecha_venta between fec_i and fec_f;

    cur_datos_precios CURSOR FOR
      select
        a.venta_id,v.fecha_venta,a.producto_id,p.nombre as nombre_articulo,
        p.marca,p.precio_proveedor,a.precio as precio_vendido,
        count(*) as cantidad
      from articulo a join producto p on a.producto_id=p.producto_id
                join venta v on a.venta_id=v.venta_id
      group by a.venta_id,v.fecha_venta,a.producto_id,
         p.nombre,p.marca,p.precio_proveedor,a.precio
      having v.fecha_venta between fec_i and fec_f;

    v_cant_total_art integer;
    v_total_pagada numeric(7,2);
    v_total_vendida numeric(7,2);
    v_ganancia numeric(7,2);
  BEGIN
    v_cant_total_art=0;
    v_total_pagada=0.0;
    v_total_vendida=0.0;

    for p in cur_datos_cant_articulos loop
      v_cant_total_art=v_cant_total_art+p.cant_total_art;
    end loop;

    for p in cur_datos_precios loop
      v_total_pagada=v_total_pagada+(p.precio_proveedor*p.cantidad);
      v_total_vendida=v_total_vendida+(p.precio_vendido*p.cantidad);
    end loop;

    v_ganancia=v_total_vendida-v_total_pagada;

    RAISE NOTICE 'Cantidad de articulos vendidos entre % y %: %',
```

```
    fec_i,fec_f,v_cant_total_art;
    RAISE NOTICE 'Total gastado: %',v_total_pagada;
    RAISE NOTICE 'Total vendido: %',v_total_vendida;
    RAISE NOTICE 'Ganancia: %',v_ganancia;
  END;
$$;
```

Ejemplo de llamada al procedimiento:

```
CALL venta_durante_temporada('2019-01-01','2019-01-31');
```

Procedimiento: venta_durante_dia(fec date) para obtener el total articulo vendidos en un día 'YYYY-MM-DD'.

```
CREATE OR REPLACE PROCEDURE venta_durante_dia(fec date)
LANGUAGE PLPGSQL
AS $$
  DECLARE
    cur_datos_cant_articulos CURSOR FOR
      select fecha_venta,sum(cantidad_articulos) as cant_total_art
      from venta group by fecha_venta
      having fecha_venta=fec;

    cur_datos_precios CURSOR FOR
      select
        a.venta_id,v.fecha_venta,a.producto_id,p.nombre as nombre_articulo,
        p.marca,p.precio_proveedor,a.precio as precio_vendido,
        count(*) as cantidad
      from articulo a join producto p on a.producto_id=p.producto_id
                join venta v on a.venta_id=v.venta_id
      group by a.venta_id,v.fecha_venta,a.producto_id,
         p.nombre,p.marca,p.precio_proveedor,a.precio
      having v.fecha_venta=fec;

    v_cant_total_art integer;
    v_total_pagada numeric(7,2);
    v_total_vendida numeric(7,2);
    v_ganancia numeric(7,2);
  BEGIN
    v_cant_total_art=0;
    v_total_pagada=0.0;
    v_total_vendida=0.0;

    for p in cur_datos_cant_articulos loop
      v_cant_total_art=v_cant_total_art+p.cant_total_art;
    end loop;

    for p in cur_datos_precios loop
```

```
      v_total_pagada=v_total_pagada+(p.precio_proveedor*p.cantidad);
      v_total_vendida=v_total_vendida+(p.precio_vendido*p.cantidad);
    end loop;

    v_ganancia=v_total_vendida-v_total_pagada;

    RAISE NOTICE
     'Cantidad de articulos vendidos el dia %: %',fec,v_cant_total_art;
    RAISE NOTICE 'Total gastado: %',v_total_pagada;
    RAISE NOTICE 'Total vendido: %',v_total_vendida;
    RAISE NOTICE 'Ganancia: %',v_ganancia;
  END;
$$;
```

Ejemplo de llamada al procedimiento:

```
CALL venta_durante_dia('2019-02-13');
```

5.5. Funcionamiento: inserción de datos

Para insertar datos a la base de datos se debe seguir un orden en específico porque, de otra forma, podrían violarse algunas restricciones de integridad de llaves foráneas. A continuación, se presenta un ejemplo de inserción de datos en el orden correcto:

```
\copy proveedor(nombre,razon_social,telefono,estado,codigo_postal,colonia,
calle,num_domicilio) from '/home/serveradmin/Downloads/datosPROVEEDOR.txt'
with NULL as 'NULL' DELIMITER '|' encoding 'UTF8';

\copy cliente(rfc,nombre,ap_paterno,ap_materno,estado,codigo_postal,colonia,
calle,num_domicilio) from '/home/serveradmin/Downloads/datosCLIENTE.txt'
with NULL as 'NULL' DELIMITER '|' encoding 'UTF8';

\copy categoria(descripcion) from
'/home/serveradmin/Downloads/datosCATEGORIA.txt'
with NULL as 'NULL' DELIMITER '|' encoding 'UTF8';

\copy correo(cliente_id,email) from
'/home/serveradmin/Downloads/datosCORREO.txt'
with NULL as 'NULL' DELIMITER '|' encoding 'UTF8';

\copy venta(venta_id,fecha_venta,pago_final,cliente_id,
cantidad_articulos) from
'/home/serveradmin/Downloads/datosVENTA.txt' with NULL as
'NULL' DELIMITER '|' encoding 'UTF8';

\copy producto(codigo_barras,nombre,precio_proveedor,fecha_compra,
marca,num_ejemplares,proveedor_id,categoria_id) from
'/home/serveradmin/Downloads/datosPRODUCTO.txt' with NULL
```

```
as 'NULL' DELIMITER '|' encoding 'UTF8';

\copy articulo(articulo_id,producto_id,precio,venta_id) from
'/home/serveradmin/Downloads/datosARTICULO.txt' with NULL as
'NULL' DELIMITER '|' encoding 'UTF8';
```

Una vez teniendo valores en las tablas, se pueden realizar inserciones a todas las tablas siempre y cuando, las inserciones a las tablas estén bien referenciadas a las llaves primarias de las tablas padre, si aplica el caso.

5.6. Funcionamiento: inserción de artículos a una venta

Para insertar un artículo se debe seguir una serie de pasos:

1. Insertar una venta primero, o referenciar el artículo a insertar a una venta que ya esté registrada. Ejemplo de inserción en venta:

   ```
   insert into venta(venta_id,fecha_venta,cliente_id)
    values('VENT-012','2019-02-14','5');
   ```

2. Realizar la siguiente consulta para saber qué producto se quiere adquirir (para completar el campo de producto_id de la tabla ARTICULO):

   ```
   select * from vst_informacion_para_registrar_articulo;
   ```

3. Para insertar artículo a venta con los debidos datos, se debe llamar al procedimiento `agregar_articulo_a_venta(producto_id,venta_id)`. Ejemplo:

   ```
   CALL agregar_articulo_a_venta(7::smallint,'VENT-010');
   ```

   Se obtendrá una advertencia si el número de ejemplares del producto es menor o igual a tres. Si el número de ejemplares del producto es igual a cero, no se realiza la transacción y se obtiene una alerta que nos indica que no existen ejemplares del producto por el momento. Para obtener nuevos ejemplares del producto, basta con hacer una actualización para la tabla PRODUCTO.

4. Para verificar el correcto funcionamiento de la insersión, hacer la siguiente consulta para comprobar que el número de ejemplares ha disminuido en una unidad del producto adquirido:

   ```
   select producto_id,num_ejemplares,nombre,marca from producto;
   ```

5. Consultar la factura de la venta con la siguiente instrucción, donde venta_id debe estar igualado al id de la venta que se acaba de actualizar:

   ```
   select * from vst_factura where venta_id='VENT-010';
   ```

6. Presentación

Conexión del formulario a la base de datos.

La solución al requerimiento planteada en el proyecto se realizó mediante una base de datos en la nube, una herramienta para crear un servidor local, y lenguaje *html*, *css* y *php*.

La base de datos está almacenada en un servidor de Clever Cloud.

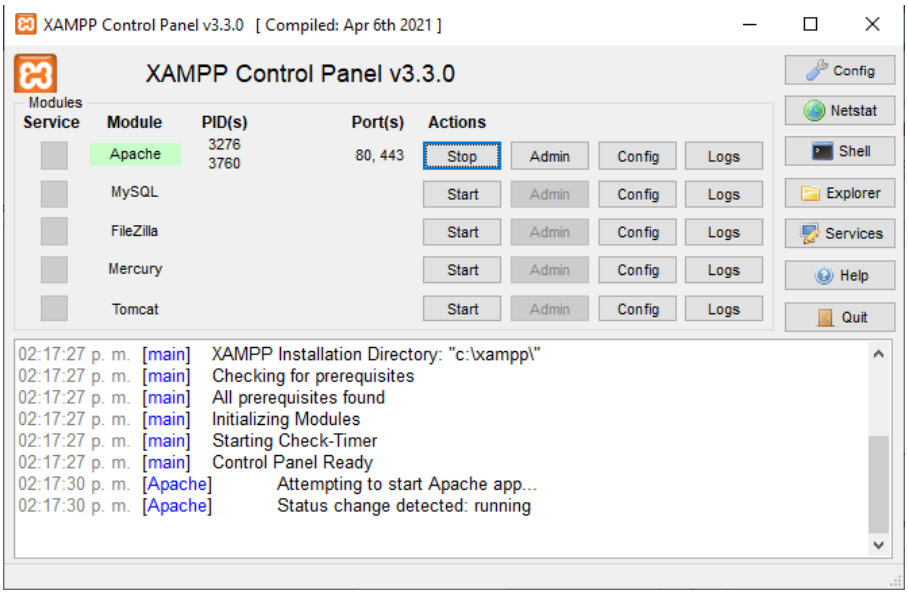
PostgreSQL by Clever Cloud — Documentation

TYPE	PLAN	CLUSTER	VERSION	REGION	STATUS	CREATED	ID
PostgreSQL	Dev	par-postgresql-c4	11	par	ACTIVE	2021-12-10	postgresql_0f35b5bf-5339-4bd9-a87a-9c8f1a480edf

De esta forma, todos los miembros del equipo pueden trabajar en la misma base de manera conjunta. Ahora bien, se debe de crear un formulario que guarde datos de un cliente y posterior a eso los ingrese a la tabla "cliente" que se encuentra en nuestra base de datos previamente hecha.

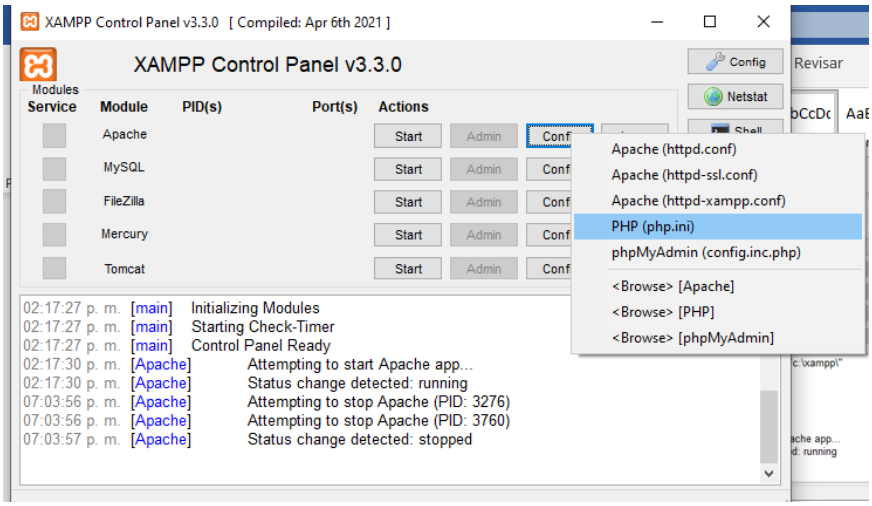
Para eso se necesitan varias herramientas que facilitaran la conexión entre las diversas partes.

Se utilizará la herramienta XAMMP:

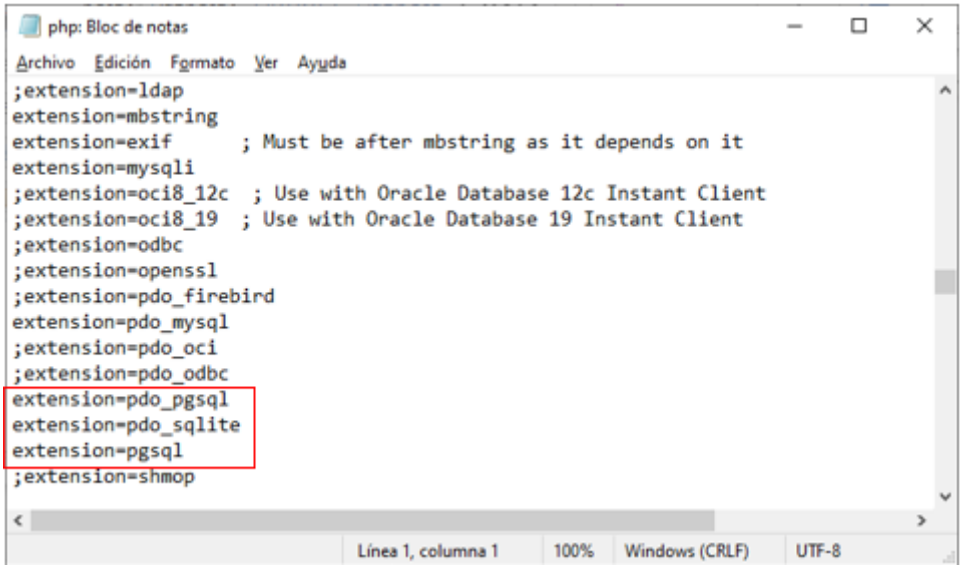


Esta herramienta permite que la computadora pueda hacer un servidor local, de esta forma podremos acceder a nuestros archivos html y php que haremos más adelante.

Para que permita la conexión a una base de datos postgres es necesario modificar unos parámetros dentro de la configuración del servicio Apache que viene en la herramienta XAMMP.



Dentro del apartado de “Dynamic Extensions” que se encuentra dentro del bloc de notas que despliega esta opción deben de retirarse los punto y coma de estas líneas que son las que corresponden a postgres.



Posterior a eso se guardan cambios y se pone en Start el apartado de Apache, para verificar que funciona correctamente como un servidor local debemos ingresar a un navegador web e ingresar “localhost/” y debería aparecer lo siguiente en nuestro explorador.

Una vez concluida esta parte podremos comenzar a trabajar en nuestro formulario html.

Su construcción se conforma de sentencias sencillas de html en las cuales irán todos los atributos que correspondan a los datos de la tabla cliente de la base de datos.

C:\xampp\htdocs\loginphp\formulario.html - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

formulario.html | enlacephp.php | prueba.php | estilo.css

```
<!DOCTYPE html>
<html>
    <head>
        <title>Formulario de Registro</title>
        <link rel ="stylesheet" type ="text/css" href="estilo.css" media="all">
    </head>
    <body>
        <div class = "form">
            <form action = "enlacephp.php" method="POST">
                <input type="text" name="RFC" placeholder="RFC"><br>
                <input type="text" name="nombre" placeholder="Nombre"><br>
                <input type="text" name="ap_paterno" placeholder="Apellido Paterno"><br>
                <input type="text" name="ap_materno" placeholder="Apellido Materno"><br>
                <input type="text" name="estado" placeholder="Estado"><br>
                <input type="text" name="codigo_postal" placeholder="Codigo Postal"><br>
                <input type="text" name="colonia" placeholder="Colonia"><br>
                <input type="text" name="calle" placeholder="Calle"><br>
                <input type="text" name="num_domicilio" placeholder="Numero de Domicilio"><br>
                <br>
                <br>
                <input type="submit" value="Enviar">
            </form>
        </div>
    </body>

</html>
```

A esto se le añadió CSS para mejorar un poco la vista del formulario, sin embargo, CSS es solo la parte estética de *html*, no afecta en el funcionamiento de nuestro archivo.

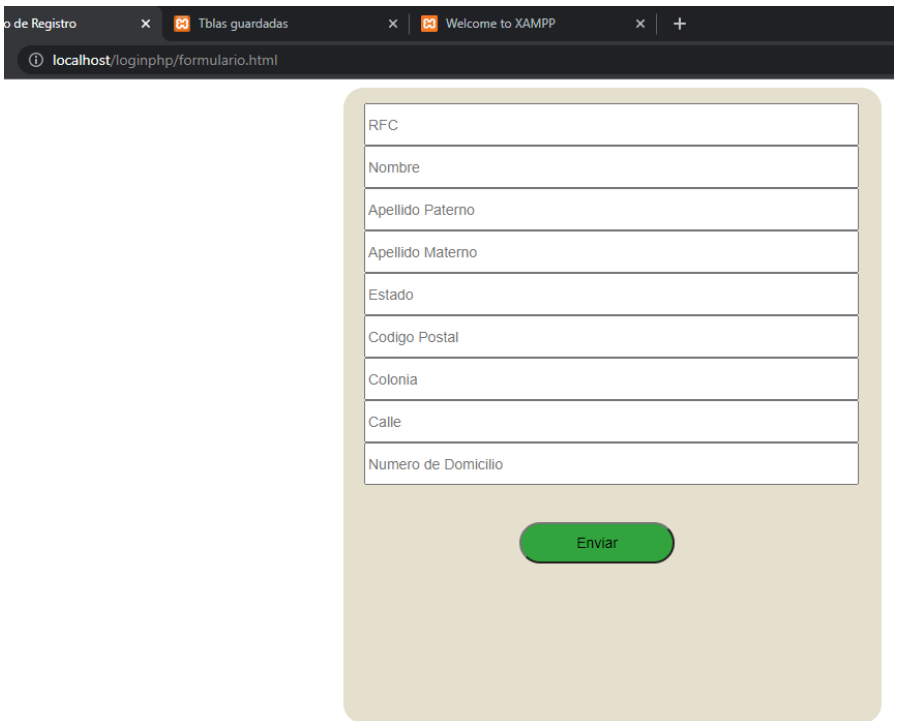
```css
.form{
    position: relative;
    width: 500px;
    height: 600px;
    padding-left: 20px;
    padding-top: 15px;
    background-color: #e5dfcd;
    border-radius: 20px;
    margin: auto;
}
input[type=text],[type=email]{
    width: 470px;
    height: 35px;
    font: 14px normal normal uppercase helvietica, arial, serif;
}

textarea{
    width: 470px;
    height: 110px;
    font: 14px normal normal uppercase helvetica, arial, serif;
}

input[type=submit]{
    position: relative;
    width: 150px;
    height: 40px;
    border-radius: 20px;
    margin-left: 150px;
    border: opx;
    background-color: #32A43E;
    font: 14px normal normal uppercase helvietica, arial, serif;

}
```

El archivo formulario.html en conjunto con estilo.css generan una vista como esta:



Ahora, para unir el formulario como la base de datos se requiere de un enlace, el cual se hará por medio de PHP, este archivo contendrá el ***string de conexión*** hacia la base de datos, además se harán sentencias SQL que permitirán ingresar los datos obtenidos del formulario hacia la tabla cliente de la base de datos alojada en la nube.

Esta parte del archivo php conecta con el servidor de la base de datos a través del string de conexión.

```php
<?php
    //conectamos Con el servidor
    $conectar=pg_connect("host=b0c9rpn69qut7zlpkr4t-postgresql.services.clever-cloud.com dbname=b0c9rpn69qut7zlpkr4t
        user=ukx2eg1y5fxtqzogudca password=Nd1MXRV2wOOrFgCXu52k");
    //verificamos la conexion
    if(!$conectar){
        echo"No Se Pudo Conectar Con El Servidor";
    }
```

Después se recuperan las variables del formulario

```php
//recuperar las variables
$RFC=$_POST['RFC'];
$nombre=$_POST['nombre'];
$ap_paterno=$_POST['ap_paterno'];
$ap_materno=$_POST['ap_materno'];
$estado=$_POST['estado'];
$codigo_postal=$_POST['codigo_postal'];
$colonia=$_POST['colonia'];
$calle=$_POST['calle'];
$num_domicilio=$_POST['num_domicilio'];
```

Se crea una sentencia SQL que realiza un insert a la tabla cliente con los valores obtenidos.

```php
//hacemos la sentencia de sql
$sql="INSERT INTO cliente (RFC, nombre, ap_paterno, ap_materno, estado, codigo_postal, colonia, calle, num_domicilio) VALUES('$RFC',
                            '$nombre',
                            '$ap_paterno',
                            '$ap_materno',
                            '$estado',
                            '$codigo_postal',
                            '$colonia',
                            '$calle',
                            '$num_domicilio')";
```

Después se ejecuta dicha sentencia y se valida la ejecución, si hay algún error en los datos o no lo permite, no se ingresarán a la tabla, caso contrario se ingresarán.

```php
    //ejecutamos la sentencia de sql
    $ejecutar=pg_query($sql);
    //verificamos la ejecucion
    if(!$ejecutar){
        echo"Hubo Algun Error<br><a href='formulario.html'>Volver</a>";
    }else{
        echo"Datos Guardados Correctamente<br><a href='formulario.html'>Volver</a>";
    }
?>
```

Podemos ver los datos insertados a la tabla haciendo uso de php, vamos a hacer que al momento de ingresar valores correctos en el formulario nos mande una tabla con los registros de los clientes, en donde deberían de verse el cliente que hayamos ingresado, de esta forma podremos visualizar la consulta sin la necesidad de realizarla en la terminal.

Datos Guardados Correctamente

Volver

ID CLIENTE	RFC	Nombre	Apellido Paterno	Apellido Materno	Estado	Codigo Postal	Colonia	Calle	Num Domicilio
1	1122334456789	JESSICA	CASTRO	ALONSO	CDMX	16500	DEL CARMEN	CALLE1	16
2	1122334456789	ENRIQUE	BALTAZAR	DOMINGUEZ	CDMX	16500	TLATELOLCO	CALLE2	9
3	1122334456789	NANCY PAOLA	PALMA	BAUTISTA	VERACRUZ	16500	AZCAPOTZALCO	CALLE3	10
4	1122334456789	MIGUEL ANGEL	ESQUIVEL	BRACAMONTE	SONORA	16500	CUAHUTEMOC	CALLE4	11
5	1122334456789	YAZMIN	ARCOS	BURGOS	VERACRUZ	16500	IZTACALCO	CALLE5	12
6	1122334456789	KARELY	MUÑOZ	CAMEZ	CDMX	16500	COYOACAN	CALLE6	13
7	1122334456789	RAFAEL	BALTAZAR	CARRASCO	CDMX	16500	XOCHIMILCO	CALLE7	14
8	1122334456789	ALEJANDRO ELIU	CANO	CASTILLO	VERACRUZ	16500	XOCHIMILCO	CALLE8	15
9	001	OIGAN	YA	PUDE	METER	1	DATO	EN LA TABLA	1
11	970127A06	JORGE	OSORIO	ALVARADO	MEXICO	57300	ALAMOS	SANTA CRUZ	73

7. Conclusiones

- Castro Alonso Jéssica
  Lo más difícil de proyecto fue hacer los procedimientos porque teníamos que hacer consultas de tal forma que obtubiéramos un dato para actualizar registros. La parte del diseño fue entre todos los miembros del equipo, por lo que fue fácil hacer aplicar los conceptos del modelo conceptual y lógico.

- Garces Gomez Eduardo Tonathiu
  El proyecto fue muy ilustrativo pues como vimos a lo largo de su elaboración se aplico todo lo que se ha visto a lo largo del curso la forma en la que se utilizaron los programas como dia, ERStudio y el manejador postgreSQL ayudaron a entender que tan útiles son para la elaboración de todo el proyecto, entendí que es muy necesaria la planificación para realizar un buen proyecto además de el tiempo pues es necesario para la realización de cada uno de los modelos que se propusieron por ejemplo el relacional y entidad-relaciona mas el ultimo pues es la base en la que se sustentara la base de datos, debemos identificar todo para que cumpla con lo solicitado y que a pesar de que se nos pueden pedir unas cosas también nosotros podemos proponer y agregar otras que nos sean de utilidad para que la base de datos se ajuste lo mas posible los requerimientos.

- Martín Alejandro Ávalos Medina
  Puedo concluir que se cumplió completamente con los objetivos, se completó la base de datos correctamente, al cursar este curso observamos cómo era paso a pso la formación de dicha base, así como aprendimos diferentes tipos de consultas y la programación interna de la base de datos para la resolución de diferentes problemas, durante este proyecto se complicó la parte de pasar a una interfaz ya la base de datos en donde se puedan ingresar los datos, en lo personal no había nunca manejado ninguna página web, por lo que se complicó un poco el lenguaje empleado para su realización, así como lograr que la base de datos se conectara correctamente con nuestro código directamente en la base de datos.

- Osorio Alvarado Jorge Adalberto
  Como conclusión respecto al proyecto en general, se pueden rescatar demasiadas cosas como tomar el tiempo necesario para crear un buen modelo relacional, ya que, si este no se hace de manera correcta, la base de datos no tendrá cierta facilidad al momento de realizar consultas, se deben de pensar muy bien la manera en que serán solicitados los datos y en que tabla vamos a colocarlos. La manera de enlazar una base de datos también tiene cierta lógica detrás, ya que de no seguir una estructura se complicará la manera de crear este enlace. Considero que de este proyecto aprendí muchas cosas, ya que a lo largo de estos semestres no había usado html, css y php. Conocerlos de manera superficial para el desarrollo de este proyecto me ha motivado a investigarlos de manera mucho mas profunda de modo que pueda seguir creando mas proyectos como este y mejorarlos tanto visualmente como en código. El campo de las bases de datos es demasiado amplio, se pueden hacer muchas cosas sabiendo implementarlas de manera correcta.

8. Bibliografía

Abatic. (2018). *PostgreSQL create Trigger*. Obtenido de todopostgresql: https://www.todopostgresql.com/postgresql-create-trigger-disparador-postgresql/

Dhumal, R. (2019). *Everything you need to know about PostgreSQL triggers*. Obtenido de enterprisedb.com: https://www.enterprisedb.com/postgres-tutorials/everything-you-need-know-about-postgresql-triggers

PostgreSQL. (2021). *ALTER SEQUENCE*. Obtenido de PostgreSQL Documentation: https://www.postgresql.org/docs/9.1/sql-altersequence.html

PostgreSQL. (2021). *ALTER TABLE*. Obtenido de PostgreSQL Documentation: https://www.postgresql.org/docs/9.1/sql-altertable.html

PostgreSQL. (2021). *COPY*. Obtenido de PostgreSQL Documentation: https://www.postgresql.org/docs/9.2/sql-copy.html

PostgreSQL. (2021). *Create Trigger*. Obtenido de PostgreSQL Documentation: https://www.postgresql.org/docs/9.1/sql-createtrigger.html

Tutorial, P. (2021). *PL/psg Cursor*. Obtenido de PostgreSQL Tutorial: https://www.postgresqltutorial.com/plpgsql-cursor/