

KDB

Innovación en papelerías

KDBCompany@gmail.com

Tel: 55 2522 5848

Cel: 55 89914793

ÍNDICE

- 1 Quiénes somos**
- 2 Objetivos**
- 3 Proyecto**
- 4 Resultados**
- 5 Conclusiones**
- 6 Sesión cliente**

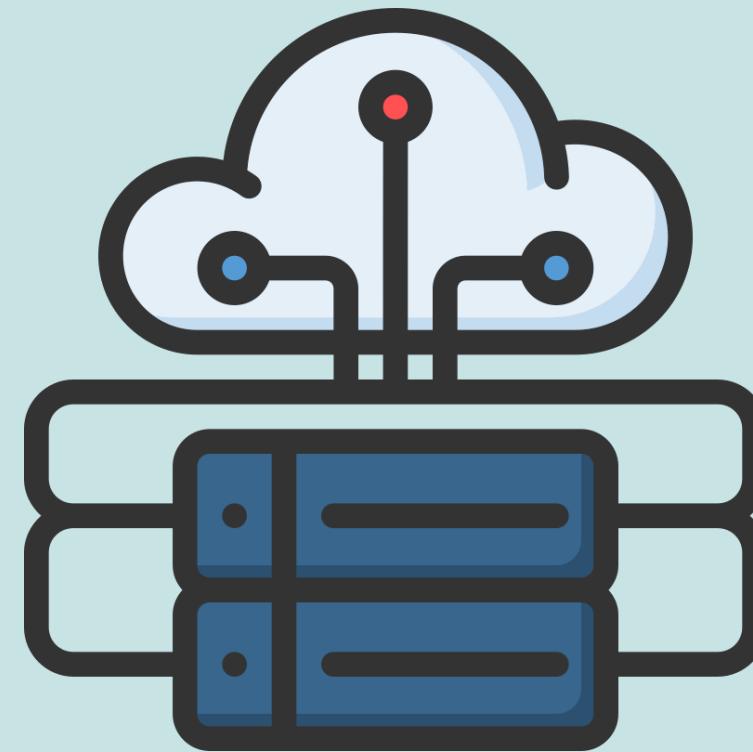


Quiénes somos

- Equipo de profesionales apasionados
- Experiencia en tecnología y gestión de proyectos
- Enfoque en soluciones innovadoras
- Desarrollo de una base de datos
- Colaboración con el sector para entender desafíos y necesidades.
- Solución personalizada para optimizar el manejo de inventario.
- Compromiso



Objetivos



Mejorar el
almacenamiento



Originalidad



Un trabajo
eficiente



El Proyecto

Consiste en el diseño de una base de datos. Una cadena de papelerías busca innovar la manera en que almacena su información, y los contratan para que desarrolle los sistemas informáticos para satisfacer los siguientes requerimientos:

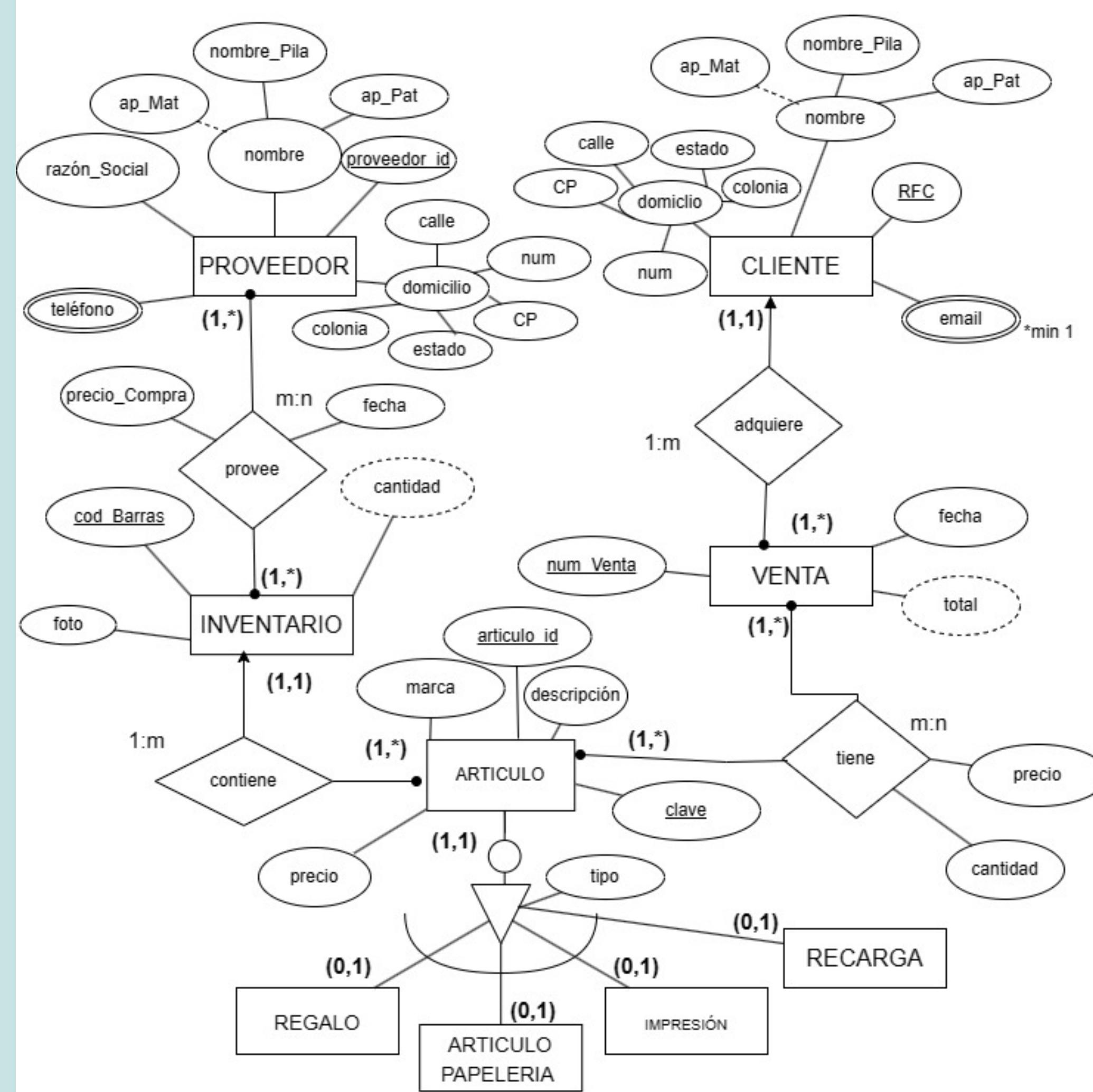
Se desea tener almacenados datos como la razón social, domicilio, nombre y teléfonos de los proveedores, rfc, nombre, domicilio y al menos un email de los clientes. Es necesario tener un inventario de los productos que se venden, en el que debe guardarse el código de barras, precio al que fue comprado el producto, foto***, 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. Adicional al almacenamiento de información, se requiere que el sistema resuelva lo siguiente:

- De manera automática se genere una vista que contenga información necesaria para asemejarse a una factura de una compra.
- Dada una fecha, o una fecha de inicio y fecha de fin, regresar la cantidad total que se vendió y la ganacia correspondiente en esa fecha/periodo.
- 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 el pedido se completa pero quedan menos de 3 en stock, se deberá emitir una alerta. Debe actualizarse el total a pagar por articulo y el total a pagar por la venta.
- Permitir obtener el nombre de aquellos productos de los cuales hay menos de 3 en stock.
- Al recibir el código de barras de un producto, regrese la utilidad.
- Crear al menos, un índice, del tipo que se prefiera y donde se prefiera. Justificar el porqué de la elección en ambos aspectos.

Resultados MER

Consiste en el diseño de una base de datos. Una cadena de papelerías busca innovar la manera en que almacena su información, y los contratan para que desarrolle los sistemas informáticos para satisfacer los siguientes requerimientos:

Se desea tener almacenados datos como la **razón social**, **domicilio**, **nombre** y **teléfonos** de los **proveedores**, **id**, **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, **foto*****, **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 **reuniones**, 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**. Adicional al almacenamiento de información, se requiere que el sistema resuelva lo siguiente:



Resultados MR

RELACIONES

- PROVEE ((id_proveedor varchar(13), cod_barrras varchar(20)) FK, PK, fecha date, precio_compra money)
- Tiene ((num_venta int, clave int) FK, PK, precio money, cantidad int)

¿Por qué este método?

- **Modelo lógico (MR)**

- **PROVEEDOR** (id_proveedor varchar(13) PK, razón_social varchar(100), nombre_proveedor varchar(70), ap_pat_proveedor varchar(50), ap_mat_proveedor varchar(50) (N), calle varchar(60), num varchar(10), cp bigint, estado varchar(70), colonia varchar(70))
- **Teléfono** (teléfono bigint (PK), id_proveedor varchar(13) FK)
- **INVENTARIO** (cod_barras varchar(20) PK, foto bytea, cantidad int (C))
- **CLIENTE** (RFC varchar(13) PK, nombre_cliente varchar(70), ap_pat_cliente varchar(50), ap_Mat_cliente varchar(50) (N), calle_cliente varchar(60), num_cliente varchar(10), cp_cliente bigint, estado_cliente varchar(70), colonia_cliente varchar(70))
- **EMAIL** (email varchar(150) (PK), RFC varchar(13) FK)
- **VENTA** (num_venta varchar(20) PK, fecha date, total money, RFC varchar(13) FK)
- **ARTICULO** (clave int PK, precio money, marca varchar(50), descripción varchar(100), cod_barras varchar(20) FK, tipo_articulo varchar(1))



Resultados

Normalización

– PROVEEDOR

PROVEEDOR	
<i>id_proveedor</i>	varchar(13) PK
<i>razón_social</i>	varchar(100)
<i>nombre_proveedor</i>	varchar(70)
<i>ap_pat_proveedor</i>	varchar(50)
<i>ap_mat_proveedor</i>	varchar(50)(N)
<i>calle</i>	varchar(60)
<i>num</i>	varchar(10)
<i>cp</i>	bigint
<i>colonia</i>	varchar(70)
<i>estado</i>	varchar(70)

DF:

id_proveedor → *razón_social*, *nombre_proveedor*, *ap_pat_proveedor*,
ap_mat_proveedor, *calle*, *num*, *cp*, *estado*, *colonia*

Normalizado
queda:

id_proveedor → *razón_social*, *nombre_proveedor*, *ap_pat_proveedor*,
ap_mat_proveedor, *calle*, *num*, *cp*
cp → *estado*, *colonia*

Las siguientes tablas cumplen 3FN:

PROVEEDOR	
id_proveedor	varchar(13) PK
razón_social	varchar(100)
nombre_proveedor	varchar(70)
ap_pat_proveedor	varchar(50)
ap_mat_proveedor	varchar(50)(N)
calle	varchar(60)
num	varchar(10)
cp	bigint FK

- Código Postal - Proveedor

CP_PROVEEDOR	
cp	bigint PK
estado	varchar(70)
colonia	varchar(70)

- TELÉFONO

TELÉFONO	
teléfono	bigint (PK)
id_proveedor	varchar(13) FK

DF:

teléfono → id_proveedor

– **INVENTARIO**

INVENTARIO	
cod_barras	varchar(20) PK
foto	bytea
cantidad	int (C)

DF:

cod_barras → foto, cantidad

– **CP_CLIENTE**

cp_cliente	bigint PK
estado_cliente	varchar(70)
colonia_cliente	varchar(70)

– **EMAIL**

EMAIL	
email	varchar(150) PK
RFC	varchar(13) FK

DF:

email → RFC

Las siguientes tablas no cumplen 3FN:

CLIENTE	
RFC	varchar(13) PK
nombre_cliente	varchar(70)
ap_pat_cliente	varchar(50)
ap_Mat_cliente	varchar(50)(N)
calle_cliente	varchar(60)
num_cliente	varchar(10)
cp_cliente	bigint
estado_cliente	varchar(70)
colonia_cliente	varchar(70)

DF:

$RFC \rightarrow nombre_cliente, ap_pat_cliente, ap_Mat_cliente,$
 $calle_cliente, num_cliente, cp_cliente, estado_cliente, colonia_cliente$

RFC → nombre_cliente, ap_pat_cliente, ap_Mat_cliente,
calle_cliente, num_cliente, cp_cliente

cp_cliente → estado_cliente, colonia_cliente

Cumple 3FN, ya que no existen dependencias transitivas.

– Cliente normalizada

CLIENTE	
RFC	varchar(13) PK
nombre_cliente	varchar(70)
ap_pat_cliente	varchar(50)
ap_Mat_cliente	varchar(50)(N)
calle_cliente	varchar(60)
num_cliente	varchar(10)
cp_cliente	bigint FK

Las siguientes tablas cumplen 3FN:

– VENTA

VENTA	
num_venta	int PK
fecha	date
total	money
RFC	varchar(13) FK

DF:

$\text{num_venta} \rightarrow \text{fecha, total, RFC}$

– ARTICULO

ARTICULO	
clave	int PK
precio	money
marca	varchar(50)
descripción	varchar(100)
cod_barras	varchar(20) FK
tipo_articulo	varchar(1)

DF:

$\text{clave} \rightarrow \text{precio, marca, descripción, cod_barras, tipo_articulo}$

– PROVEE

PROVEE	
(id_proveedor, cod_barras)	(varchar (13), varchar (20)) PK FK
fecha	date
precio_compra	money

DF:

$(\text{id_proveedor}, \text{cod_barras}) \rightarrow \text{fecha, precio_compra}$

– TIENE

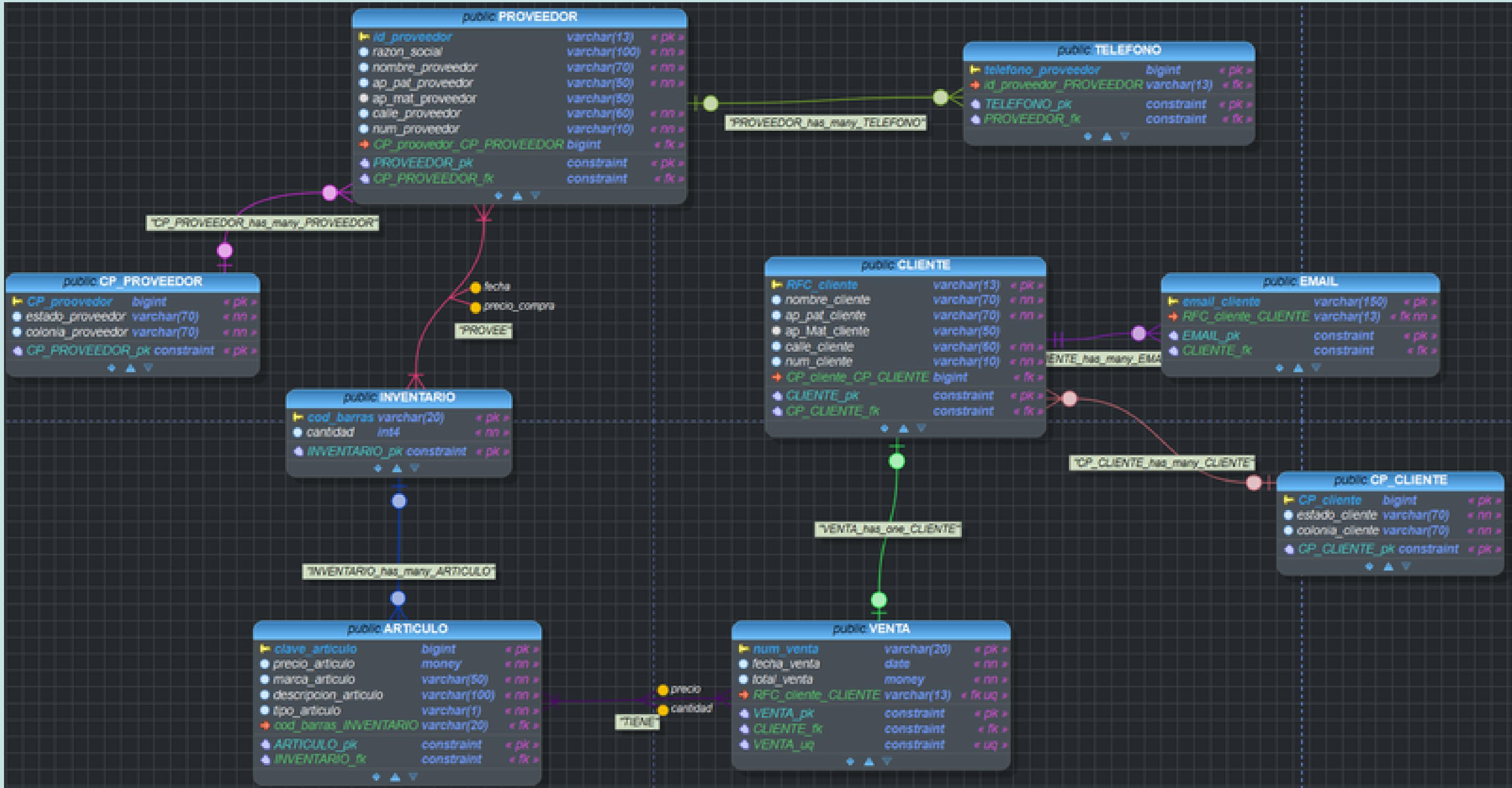
TIENE	
(num_venta, clave)	(int, int) PK FK
precio	money
cantidad	int

DF:

$(\text{num_venta}, \text{clave}) \rightarrow \text{precio, cantidad}$

Resultados

Modelo Físico



Resultados Consultas

\i nombre_archivo.sql

Para cargar archivos

```
-- Creación vista
CREATE VIEW vista_factura AS
SELECT
    V.num_venta AS num_factura,
    V.fecha_venta AS fecha,
    C.RFC_cliente AS rfc_cliente,
    C.nombre_cliente AS nombre_cliente,
    C.ap_pat_cliente || ' ' || C.ap_Mat_cliente AS apellidos_cliente,
    C.calle_cliente || ', ' || C.num_cliente AS direccion_cliente,
    CP.estado_cliente AS estado_cliente,
    CP.colonia_cliente AS colonia_cliente,
    A.clave_articulo AS clave_articulo,
    A.descripcion_articulo AS descripcion_articulo,
    A.marca_articulo AS marca_articulo,
    T.cantidad AS cantidad,
    A.precio_articulo AS precio_unitario,
    T.precio * T.cantidad AS precio_total
FROM
    public.venta V
    JOIN public.CLIENTE C ON V.RFC_cliente_CLIENTE = C.RFC_cliente
    JOIN public.CP_CLIENTE CP ON C.CP_cliente_CP_CLIENTE = CP.CP_cliente
    JOIN public.TIENE T ON V.num_venta = T.num_venta_VENTA
    JOIN public.ARTICULO A ON T.clave_articulo_ARTICULO = A.clave_articulo;
```

```
-- Dado una fecha de inicio y una de fin, mostrar la ganancia total y la cantidad total
SELECT
    SUM(T.cantidad) AS Cantidad_total,
    SUM((T.cantidad * A.precio_articulo) - (P.precio_compra * T.cantidad)) AS Ganancia_total
FROM
    public.TIENE T
    JOIN public.ARTICULO A ON T.clave_articulo_ARTICULO = A.clave_articulo
    JOIN public.PROVEE P ON A.cod_barras_INVENTARIO = P.cod_barras_INVENTARIO
    JOIN public.VENTA V ON T.num_venta_VENTA = V.num_venta
WHERE
    V.fecha_venta >= '2023-01-01' AND V.fecha_venta <= '2023-06-30';
```

```
--Creacion trigger para actualizar el inventario
CREATE OR REPLACE FUNCTION actualizar_inventario()
RETURNS TRIGGER AS $$

DECLARE
producto_id INT;
cant_actual INTEGER;
BEGIN

-- Obtener los valores del registro insertado
producto_id := NEW.clave_articulo_articulo;
cant_actual := NEW.cantidad;

-- Actualizar el valor de cantidad de INVENTARIO
UPDATE inventario
SET cantidad = cantidad - NEW.cantidad
WHERE cod_barras = (SELECT cod_barras_INVENTARIO FROM ARTICULO WHERE clave_articulo = producto_id);

-- Verificar si el stock llega a cero para abortar la transacción
IF (SELECT cantidad FROM inventario WHERE cod_barras = (SELECT cod_barras_INVENTARIO FROM ARTICULO WHERE clave_articulo = producto_id)) <= 0 THEN
    RAISE EXCEPTION 'El stock ha llegado a cero. La transacción ha sido abortada.';
    RETURN NULL;
END IF;

-- Emitir alerta si el stock es menor o igual a 3
IF (SELECT cantidad FROM inventario WHERE cod_barras = (SELECT cod_barras_INVENTARIO FROM ARTICULO WHERE clave_articulo = producto_id)) <= 3 THEN
    RAISE NOTICE '¡Alerta! Quedan pocas unidades en stock.';
END IF;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
-- Nombre y stock de los articulos con menos de 3 de stock
SELECT a.descripcion_articulo, i.cantidad
FROM ARTICULO a
INNER JOIN INVENTARIO i ON a.cod_barras_INVENTARIO = i.cod_barras
WHERE i.cantidad < 3;
-- No sale nada porque aun no tenemos articulos con menos de 3 de stock
```

```
--Al recibir el codigo de barras mostrar la utilidad
SELECT V.num_venta, ((A.precio_articulo - P.precio_compra) * T.cantidad) AS utilidad
FROM TIENE T
JOIN ARTICULO A ON T.clave_articulo_articulo = A.clave_articulo
JOIN PROVEE P ON A.cod_barras_INVENTARIO = P.cod_barras_INVENTARIO
JOIN venta v ON T.num_venta_venta = V.num_venta
WHERE A.cod_barras_INVENTARIO = '123456789012';
```

```
-- Creación de indice
CREATE INDEX idx_marca_articulo ON public.ARTICULO (marca_articulo);
```

Conclusion

Lo que logramos

Dificultades encontradas

Futuras mejoras



Gracias

ESPERAMOS SU LLAMADA

Sesión Cliente

