Proyecto Final

Badillo Aguilar Diego Jiménez Hernández Diana
Labastida Vázquez Fernando Salgado Valdés Andrés

10 de Junio, 2023

Introducción: Análisis del problema

El problema consiste en diseñar una base de datos para una cadena de papelerías que busca innovar la forma en que almacena su información. Se requiere desarrollar sistemas informáticos que cumplan con los siguientes requerimientos:

- Almacenamiento de datos de proveedores: Se deben almacenar datos como la razón social, domicilio, nombre y teléfonos de los proveedores. Además, se debe incluir el Registro Federal de Contribuyentes (RFC) de cada proveedor.
- Almacenamiento de datos de clientes: Se debe almacenar información de los clientes, como su nombre, domicilio y al menos un correo electrónico.
- Inventario de productos: Es necesario tener un inventario de los productos que se venden en la papelería. Para cada producto, se debe almacenar el código de barras, el precio al que fue comprado, una foto, la fecha de compra y la cantidad de ejemplares en la bodega (stock).
- Regalos, artículos de papelería, impresiones y recargas: Se deben almacenar datos de los regalos, artículos de papelería, impresiones y recargas que se venden en la papelería. Sin embargo, estos datos solo deben ser guardados si existe un registro correspondiente en el inventario de productos. Para cada artículo, se deben guardar la marca, descripción y precio.
- Datos de ventas: Es necesario almacenar información de cada venta realizada en la papelería. Para cada venta, se debe guardar el número de venta, la fecha de venta y la cantidad total a pagar. Además, se debe almacenar la cantidad de cada artículo vendido y el precio total a pagar por cada artículo.

Plan de trabajo

El plan de trabajo para la elaboración del proyecto de creación de la base de datos se dividió en varias etapas, asignadas a diferentes miembros del equipo. A continuación, se detalla el desglose de tareas y responsabilidades:

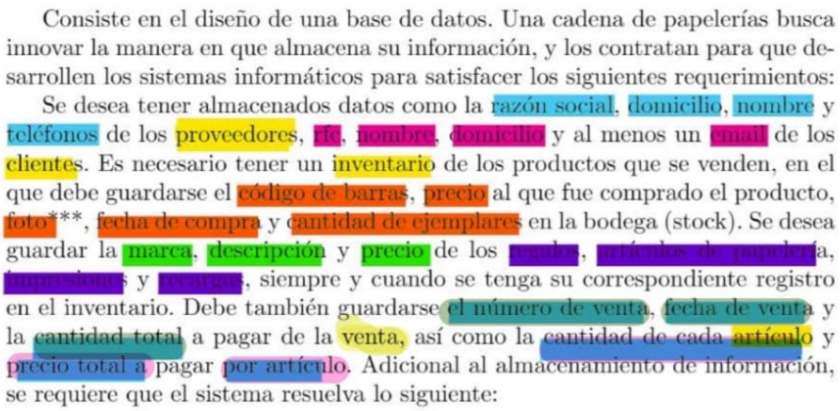
1. Análisis de requerimientos y modelo conceptual: Diana Jiménez Hernández
2. Modelo lógico: Andrés Salgado Valdés
3. Normalización: Fernando Labastida Vázquez
4. Creación del modelo físico: Diego Badillo Aguilar
5. Elaboración, pruebas y creación de datos de prueba: Todos los miembros del equipo
6. Creación del documento: Diego Badillo Aguilar, Labastida Vázquez Fernando

Detalles de cada etapa:

1. **Análisis de requerimientos y modelo conceptual:** Recopilar los requerimientos del proyecto y elaborar el modelo conceptual de la base de datos.
2. **Modelo lógico:** Desarrollar el modelo lógico de la base de datos, utilizando como referencia el modelo conceptual creado por Diana.
3. **Normalización:** Realizar el proceso de normalización de la base de datos, asegurando la integridad de la base de datos y reduciendo la redundancia de los datos.
4. **Creación del modelo físico):** Crear el modelo físico de la base de datos, basándose en el modelo lógico previamente desarrollado por Andrés.
5. **Elaboración, pruebas y creación de datos de prueba:** Todos los miembros del equipo colaborarán conjuntamente en esta fase, trabajando de manera colaborativa para elaborar y lograr los requerimientos a nivel PL/SQL, así como realizar las pruebas necesarias para garantizar la calidad y funcionalidad de la base de datos. Además, se crearán datos de prueba para validar el correcto funcionamiento del sistema.
6. **Creación del documento:** Redactar y elaborar el documento final del proyecto, donde se detallarán todos los aspectos relevantes del proceso de creación de la base de datos.

Diseño

- Análisis de requerimientos



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 desarrollen 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:

Figure 1: Análisis del enunciado con los requerimientos

  - **Entidades**
    - Las entidades obtenidas fueron: proveedores, clientes e inventario. Están resaltadas de color amarillo.
    - También nos dimos cuenta de las siguientes supertipos: regalos, artículos de papelería, impresiones y recargas. Están resaltadas de color morado.
  - **Atributos de las entidades**
    - Entidad proveedores
      - razón social, domicilio, nombre, teléfonos. Estos están resaltados en color Azul
    - Entidad cliente
      - RFC, nombre, domicilio y email
    - Entidad inventario
      - Código de barras, precio, foto, fecha de compra y cantidad de ejemplares
    - Entidad articulo
      - Marca, descripción y precio
    - Entidad venta
      - El numero de venta, fecha de venta y cantidad total.
  - **Relaciones**
    - provee, adquiere, tiene (la cual tiene de atributos: cantidad de cada articulo, precio total, articulo)

- Modelo conceptual

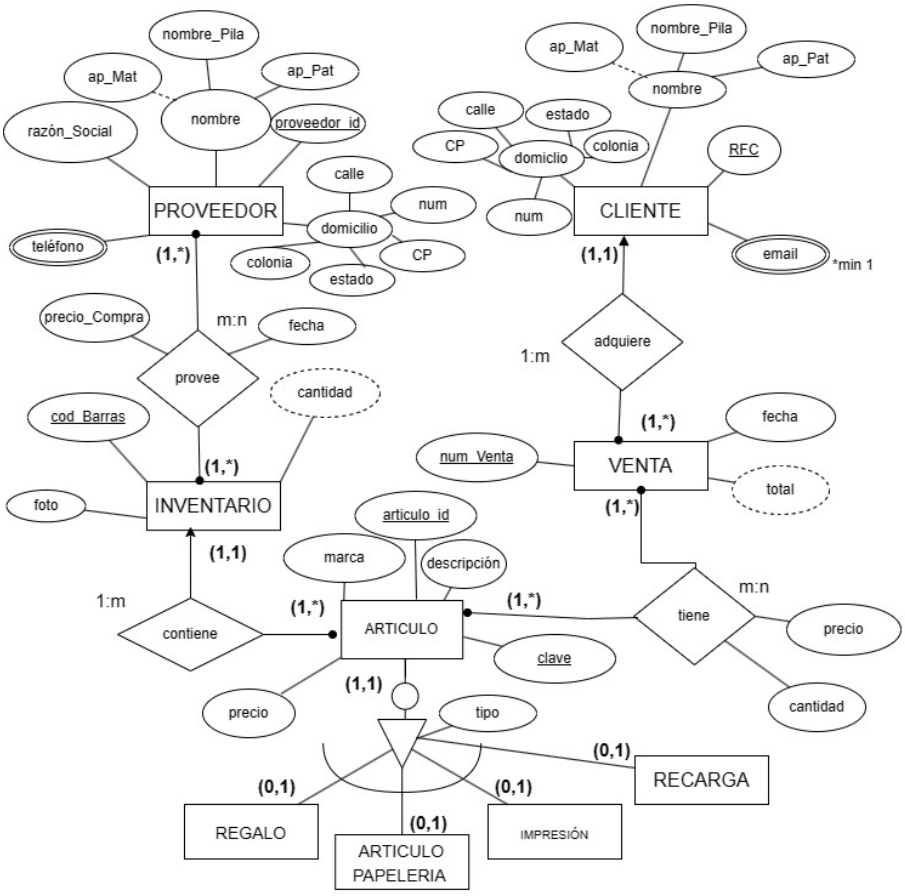


Figure 2: Modelo conceptual

Se identifica que existe un supertipo ya que las entidades recarga, impresión, artículo de papelería y regalo comparten los mismos atributos. Se presenta una exclusión ya que por lógica uno de estos artículos no puede ser dos de estas cosas a la vez y es parcial ya que pueden existir artículos que no pertenezcan a ninguna de estas. Se presenta una relación entre entre venta y artículo ya que se deben registrar la cantidad y precio total por artículo. También se presenta una relación entre artículo e inventario ya que se indica que se indica que para primero debe existir su registro en el inventario. Existe una relación entre proveedor e inventario de la cual se requiere almacenar el precio de compra y la fecha. Además, existe la relación entre cliente y venta la cuál se intuye por el contexto del problema. Existen dos atributos multivaluados los teléfonos de proveedor que se indica de manera explícita y los emails del cliente porque se indica

que debe registrarse mínimo uno, por lo que puede presentarse varios. Adicionalmente el atributo cantidad de la entidad inventario se considera calculado debido a que su valor depende de la existencia en stock en relación a la adquisición de productos y las ventas. De la misma forma, el atributo total de la venta se considera calculado porque depende del precio total por artículo.

- **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))

  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?
  - Podemos ver que los subtipos de la tabla ARTICULO no tienen atributos adicionales y existe una exclusión parcial entre ellos. La solución más adecuada sería utilizar una relación de exclusión en lugar de tablas separadas para los subtipos.
  - Con esta estructura podemos ver que el atributo tipo_articulo servirá para saber a qué subtipo pertenece: pondremos R de regalo, P de papelería, I de impresión y R de recarga.

- En cuanto a las relaciones PROVEE y TIENE, al ser muchos a muchos, se crea una nueva relación que tendrá como PK las PKs de las entidades que une (que a su vez pasarán como FKs), más los atributos de la relación.
- Para las relaciones de 1 a muchos o muchos a 1, debemos llevar la PK de la cardinalidad 1 como FK a la de cardinalidad M.

- Normalización
  - **PROVEEDOR**

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
colonia	varchar(70)
estado	varchar(70)

DF:

$$\begin{aligned} \textit{id\_proveedor} \rightarrow\ & \text{razón\_social, nombre\_proveedor, ap\_pat\_proveedor,} \\ & \text{ap\_mat\_proveedor, calle, num, cp, estado, colonia} \end{aligned}$$

Cumple 1FN, ya que no existen atributos multivaluados, no hay grupos de repetición y tiene PK.

Cumple 2FN, ya que no existen dependencias parciales.

No cumple 3FN porque con cp se puede encontrar a estado y a colonia.

Entonces, ya normalizada quedaría:

$$\begin{aligned} \textit{id\_proveedor} \rightarrow\ & \text{razón\_social, nombre\_proveedor, ap\_pat\_proveedor,} \\ & \text{ap\_mat\_proveedor, calle, num, cp} \\ \textit{cp} \rightarrow\ & \text{estado, colonia} \end{aligned}$$

Cumple 3FN, ya que no existen dependencias transitivas.

Las tablas cumpliendo 3FN son:

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:

$$\text{teléfono} \rightarrow \text{id\_proveedor}$$

Cumple 1FN, ya que no existen atributos multivaluados, no hay grupos de repetición y tiene PK.

Cumple 2FN, ya que no existen dependencias parciales.

Cumple 3FN, ya que no existen dependencias transitivas.

- **INVENTARIO**

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

DF:

$$\text{cod\_barras} \rightarrow \text{foto, cantidad}$$

Cumple 1FN, ya que no existen atributos multivaluados, no hay grupos de repetición y tiene PK.

Cumple 2FN, ya que no existen dependencias parciales.

Cumple 3FN, ya que no existen dependencias transitivas.

- **CLIENTE**

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

Cumple 1FN, ya que no existen atributos multivaluados, no hay grupos de repetición y tiene PK.

Cumple 2FN, ya que no existen dependencias parciales.

Esta dependencia funcional no cumple 3FN ya que con cp_cliente se puede encontrar a estado_cliente y a colonia_cliente, por lo tanto hay transitividad. Las relaciones normalizadas quedarían de la siguiente manera:

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

$cp\_cliente \rightarrow$ 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

- **Código Postal_Cliente**

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:

$$\text{email} \rightarrow \text{RFC}$$

Cumple 1FN, ya que no existen atributos multivaluados, no hay grupos de repetición y tiene PK.

Cumple 2FN, ya que no existen dependencias parciales.

Cumple 3FN, ya que no existen dependencias transitivas.

- **VENTA**

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

DF:

$$\text{num\_venta} \rightarrow \text{fecha, total, RFC}$$

Cumple 1FN, ya que no existen atributos multivaluados, no hay grupos de repetición y tiene PK.

Cumple 2FN, ya que no existen dependencias parciales.

Cumple 3FN, ya que no existen dependencias transitivas.

- **ARTICULO**

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

DF:

$$\text{clave} \rightarrow \text{precio, marca, descripción, cod\_barras, tipo\_articulo}$$

Cumple 1FN, ya que no existen atributos multivaluados, no hay grupos de repetición y tiene PK.

Cumple 2FN, ya que no existen dependencias parciales.

Cumple 3FN, ya que no existen dependencias transitivas.

- **PROVEE**

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

DF:

$$(id\_proveedor, cod\_barras) \rightarrow \text{fecha, precio\_compra}$$

Cumple 1FN, ya que no existen atributos multivaluados, no hay grupos de repetición y tiene PK.

Cumple 2FN, ya que no existen dependencias parciales.

Cumple 3FN, ya que no existen dependencias transitivas.

- **TIENE**

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

DF:

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

Cumple 1FN, ya que no existen atributos multivaluados, no hay grupos de repetición y tiene PK.

Cumple 2FN, ya que no existen dependencias parciales.

Cumple 3FN, ya que no existen dependencias transitivas.

Implementación(Modelo físico)

Para la implementación del modelo físico se opto en primera instancia en crear primero el diagrama de nuestra base de datos ya normalizada en pgModeler para lograr así reducir errores humanos y agilizar la creación.

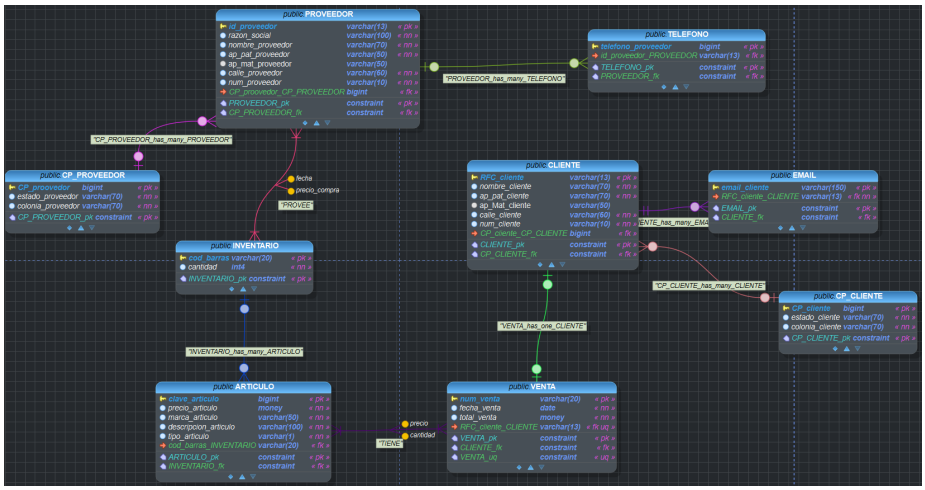


Figure 3: Diagrama físico hecho en pgModeler

Una vez creado logramos obtener el código que esta en el documento "Proyecto.sql" En el cual estará la creación de la base de datos papelería,inserciones de datos para poder comprobar la tabla y la creación de las funciones y triggers para poder cumplir los requerimientos. A continuación se explicara de manera detallada como se llego a la resolución de cada uno de los requerimientos.

1. **Carga de archivos:** Para cargar los archivos en postgres primero debes ingresar en la terminal y mediante el comando cd, debes moverte dentro de tus carpetas y archivos hasta llegar al directorio donde se encuentra Proyecto.sql

   Una vez estés allí, escribe la palabra 'psql' (sin ') para abrir postgres e iniciar sesión. Ahora solo tienes que escribir

   ```
   \i nombre_achivo.sql
   ```

   Y listo, verás como se crean los objetos, se insertan los datos y se crean los triggers, funciones y procedimientos empleados en nuestra base de datos.

   Para comprobar que todo salió bien, teclea 'dt' para ver los objetos existentes en tu base de datos. Y si deseas comprobar que también los datos fueron insertados, escribe:

   ```
   select * from nombre_tabla
   ```

2. **De manera automática se genere una vista que contenga información necesaria para asemejarse a una factura de una compra:**

```sql
-- 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 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;
```

Figure 4: Código de la creación de la vista factura

(a) El código comienza creando una vista llamada `vista_factura` utilizando la sentencia `CREATE VIEW`. Una vista es una tabla virtual que se basa en los resultados de una consulta.

(b) La consulta principal de la vista se define con la sentencia `SELECT`.

(c) La consulta selecciona varias columnas de diferentes tablas y las renombra según convenga.

(d) En la primera línea de la consulta, la columna `num_venta` de la tabla `V` se renombra como `num_factura` en la vista.

(e) En la segunda línea, la columna `fecha_venta` de la tabla `V` se renombra como `fecha` en la vista.

(f) En la tercera y cuarta líneas, las columnas `RFC_cliente` y `nombre_cliente` de la tabla `C` se renombran como `rfc_cliente` y `nombre_cliente` respectivamente en la vista.

(g) En la quinta línea, se utiliza concatenación de cadenas para combinar las columnas `ap_pat_cliente` y `ap_Mat_cliente` de la tabla `C`. El resultado se renombra como `apellidos_cliente` en la vista.

(h) En la sexta línea, se utiliza concatenación de cadenas para combinar las columnas `calle_cliente` y `num_cliente` de la tabla `C`. El resultado se renombra como `direccion_cliente` en la vista.

(i) En la séptima y octava líneas, las columnas `estado_cliente` y `colonia_cliente` de la tabla `CP` se renombran como `estado_cliente` y `colonia_cliente` respectivamente en la vista.

(j) En la novena línea, la columna `clave_articulo` de la tabla `A` se renombra como `clave_articulo` en la vista.

(k) En la décima línea, la columna `descripcion_articulo` de la tabla `A` se renombra como `descripcion_articulo` en la vista.

(l) En la undécima línea, la columna `marca_articulo` de la tabla `A` se renombra como `marca_articulo` en la vista.

(m) En la duodécima línea, la columna `cantidad` de la tabla `T` se renombra como `cantidad` en la vista.

(n) En la decimotercera línea, la columna `precio_articulo` de la tabla `A` se renombra como `precio_unitario` en la vista.

(o) En la decimocuarta línea, la columna `precio` de la tabla `T` se renombra como `precio_total` en la vista.

(p) La cláusula `FROM` indica las tablas que se utilizarán en la consulta.

(q) En la línea 15, se realiza una operación de unión (`JOIN`) entre las tablas `VENTA` y `CLIENTE` utilizando las columnas `RFC_cliente_CLIENTE` de `V` y `RFC_cliente` de `C` respectivamente.

(r) En la línea 16, se realiza una operación de unión (`JOIN`) entre las tablas `CLIENTE` y `CP_CLIENTE` utilizando las columnas `CP_cliente_CP_CLIENTE` de `C` y `CP_cliente` de `CP` respectivamente.

(s) En la línea 17, se realiza una operación de unión (`JOIN`) entre las tablas `VENTA` y `TIENE` utilizando las columnas `num_venta` de `V` y `num_venta_VENTA` de `T` respectivamente.

(t) En la línea 18, se realiza una operación de unión (`JOIN`) entre las tablas `TIENE` y `ARTICULO` utilizando las columnas `clave_articulo_ARTICULO` de `T` y `clave_articulo` de `A` respectivamente.

(u) Finalmente, el punto y coma indica el final de la consulta y el código.

En resumen, el código crea una vista llamada `vista_factura` que combina información de varias tablas: `VENTA`, `CLIENTE`, `CP_CLIENTE`, `TIENE` y `ARTICULO`. La vista selecciona columnas específicas de cada tabla y les asigna nuevos nombres. Algunas columnas se concatenan utilizando operaciones de cadena. La finalidad de esta vista es proporcionar una estructura simplificada y conveniente para consultar información relacionada con facturas.

3. **Dada una fecha, o una fecha de inicio y fecha de fin, regresar la cantidad total que se vendió y la ganancia correspondiente en esa fecha/periodo.:**

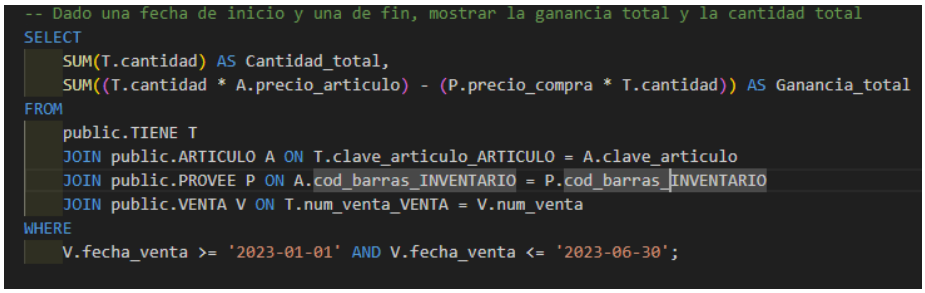


Figure 5: Creación de la consulta para obtener la ganancia por fecha

El código realiza una consulta para calcular la cantidad total y la ganancia total obtenida en ventas durante un período de tiempo específico.

En la cláusula `SELECT`, se utiliza la función de agregación `SUM()` para calcular la suma de la cantidad vendida (`T.cantidad`) y la ganancia obtenida por cada venta. Estos valores se nombran como `Cantidad_total` y `Ganancia_total`, respectivamente.

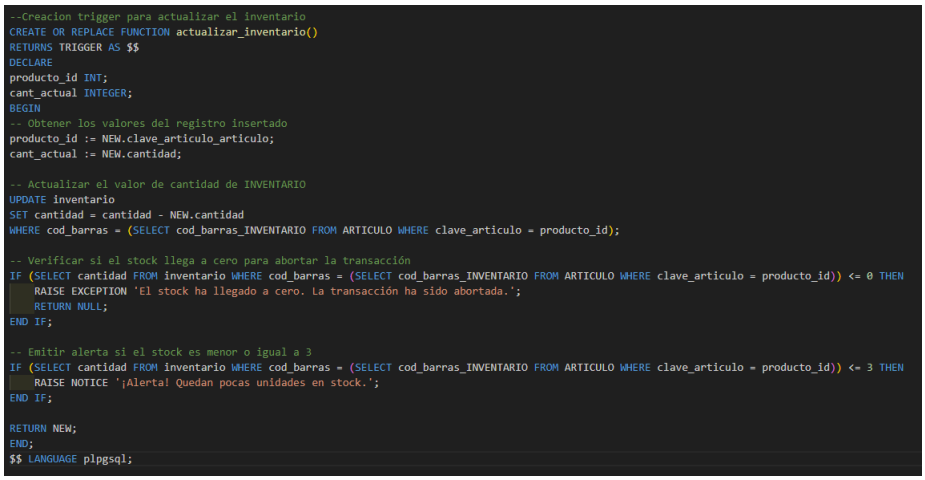
En la cláusula `FROM`, se especifican las tablas involucradas en la consulta y se les asignan alias. Las tablas utilizadas son `TIENE`, `ARTICULO`, `PROVEE` y `VENTA`, que se abrevian como `T`, `A`, `P` y `V`, respectivamente.

La cláusula `JOIN` se utiliza para combinar las tablas relacionadas. En este caso, se unen las tablas `TIENE` y `ARTICULO` utilizando la condición de igualdad `T.clave_articulo_ARTICULO = A.clave_articulo`. Luego, se une la tabla `ARTICULO` con la tabla `PROVEE` utilizando la condición `A.cod_barras_INVENTARIO = P.cod_barras_INVENTARIO`. Finalmente, se une la tabla `TIENE` con la tabla `VENTA` utilizando la condición `T.num_venta_VENTA = V.num_venta`.

En la cláusula `WHERE`, se establecen las condiciones para filtrar los datos. En este caso, se especifica que solo se deben considerar las ventas que ocurrieron dentro del período de tiempo del 1 de enero de 2023 al 30 de junio de 2023. Esto se logra utilizando las condiciones `V.fecha_venta >= '2023-01-01'` y `V.fecha_venta <= '2023-06-30'`.

En resumen, la consulta calcula la cantidad total vendida y la ganancia total obtenida durante un período de tiempo específico. Combina varias tablas relacionadas y utiliza condiciones para filtrar los datos según la fecha de venta.

4. **Cada que haya la venta de un articulo, deberá decrementarse el stock por la cantidad vendida de ese articulo. 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.:**



```sql
--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;
```

Figure 6: Código para la creación del trigger que controle las cantidades en inventario

Este código en SQL crea una función y un desencadenador (trigger) en la base de datos.

La función se llama `actualizar_inventario` y está definida con la cláusula `CREATE OR REPLACE FUNCTION`. Esta función tiene un tipo de retorno de `TRIGGER`, lo que significa que será utilizada como un trigger (desencadenador) para ejecutar ciertas acciones cuando ocurra un evento específico en la base de datos.

Dentro de la función, hay declaraciones de variables como `producto_id` y `cant_actual`. Estas variables se utilizan para almacenar los valores del registro insertado en la tabla que activa el trigger.

A continuación, se realiza una actualización en la tabla `inventario` utilizando la cláusula `UPDATE`. La cantidad en la tabla se actualiza restando la cantidad del nuevo registro insertado. El filtro en la cláusula `WHERE` se basa en la coincidencia de los códigos de barras entre las tablas `inventario` y `articulo`.

Después de la actualización, se realiza una verificación para ver si el stock ha llegado a cero. Si la cantidad en la tabla `inventario` es menor o igual a cero, se genera una excepción usando la cláusula `RAISE EXCEPTION` y se aborta la transacción.

Luego, se emite una alerta si el stock es menor o igual a 3 unidades utilizando la cláusula `RAISE NOTICE`.

Finalmente, la función devuelve el nuevo registro insertado utilizando `RETURN NEW`.

El segundo bloque de código `CREATE TRIGGER` crea un trigger llamado `trigger_actualizar_inventario` en la tabla `TIENE`. Este trigger se activa

después de cada inserción (`AFTER INSERT`) en la tabla `TIENE` y ejecuta la función `actualizar_inventario`.

En resumen, este código crea un desencadenador que se activará después de cada inserción en la tabla `TIENE` y ejecutará la función `actualizar_inventario`. La función actualiza la cantidad en la tabla `inventario`, verifica si el stock llega a cero, emite una alerta si el stock es bajo y devuelve el nuevo registro insertado.

5. **Al recibir el código de barras de un producto, regrese la utilidad.**

```sql
--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';
```

Figure 7: Código para obtener la utilidad con el código de barras

La consulta busca mostrar la utilidad obtenida en una venta específica para un artículo en particular. El usuario puede elegir el valor del campo `A.cod_barras_INVENTARIO` para obtener la utilidad de un artículo específico.

En la cláusula `WHERE`, se especifica `A.cod_barras_INVENTARIO = '123456789012'`, lo cual indica que se está buscando la utilidad para el artículo cuyo código de barras en la tabla `ARTICULO` sea igual a '123456789012'. Este valor '123456789012' es un ejemplo, y el usuario puede modificarlo para seleccionar el código de barras de un artículo específico.

La consulta realiza múltiples `JOIN` con las tablas `TIENE`, `ARTICULO`, `PROVEE` y `VENTA` para obtener los datos necesarios. En resumen, la consulta une las tablas utilizando las correspondientes claves foráneas y selecciona el número de venta (`V.num_venta`) y la utilidad calculada (`((A.precio_articulo - P.precio_compra) * T.cantidad)`) para el artículo con el código de barras especificado.

En conclusión, el usuario puede seleccionar cualquier dato de código de barras válido en el campo `A.cod_barras_INVENTARIO` en la cláusula `WHERE` para obtener la utilidad de venta de un artículo específico. Simplemente deben reemplazar el valor '123456789012' en la consulta con el código de barras deseado.

6. **Crear al menos, un índice, del tipo que se prefiera y donde se prefiera. Justificar el porque de la elección en ambos aspectos:**

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

Figure 8: Código para la creación del índice

Se utilizó un índice en la tabla `artículo` en el atributo `marca` para tener un control de los artículos de acuerdo a su marca con el propósito de que sea más fácil identificar a qué proveedor se requiere llamar en caso de no necesitar reabastecer el artículo. De esta forma, es más sencillo obtener la lista de artículos que corresponde a cada proveedor en lugar de llamar cada vez que se identifique artículo por artículo, ya que su búsqueda se vuelve más eficiente.

Se eligió un tipo de índice no agrupado (*non-clustered*) debido a que de esta forma los artículos se ordenan por la marca, logrando así su función.

Conclusiones

1. Lo que logramos:

   - Labastida Vázquez Fernando

     Creamos una base de datos adecuada a la situación: La diseñamos y programamos de manera que cumpliera con todo lo que necesitábamos para este proyecto. Queríamos que fuera eficiente y pudiera manejar mucha información sin problemas.

   - Badillo Aguilar Diego

     Aseguramos que los datos estuvieran bien conectados: Nos aseguramos de que las diferentes partes de la base de datos se relacionaran entre sí de la forma correcta. También pusimos algunas reglas para que los datos siempre estuvieran en buen estado y no hubiera errores.

   - Jiménez Hernández Diana:

     Hicimos que el inventario se actualizara automáticamente: Programamos una función que se ejecutaba cada vez que se agregaba algo nuevo a la tabla "TIENE". Así, el inventario se mantenía actualizado sin tener que hacerlo manualmente.

   - Salgado Valdés Andrés

     Aprendimos a cargar datos desde archivos: Aprendimos a usar un comando llamado "\i" para cargar grandes cantidades de datos desde archivos externos. Esto nos ahorró mucho tiempo y esfuerzo.

2. Las dificultades que encontramos:

   - Salgado Valdés Andrés

Al principio, nos costó un poco diseñar la base de datos: Tuvimos que tomar decisiones difíciles y hacer algunos ajustes para asegurarnos de que todo estuviera bien organizado. No queríamos que hubiera información repetida o inconsistente.

- Badillo Aguilar Diego

  La programación de funciones y disparadores fue un poco complicada: Tuvimos que aprender un lenguaje de programación específico para hacer esto, y al principio fue confuso. Pero con práctica y estudio, pudimos hacerlo bien.

- Labastida Vázquez Fernando

  Postgres en múltiples ocasiones presentaba errores que no debía de presentar. Por ejemplo, a la hora de crear una vista, por algún motivo no devolvía nada, sólo los encabezados de la tabla, pero sin información. Comprobamos si la consulta estaba bien escrita, y así era. Lo solucionamos simplemente saliendo de la base de datos y volviendo a entrar, y ya así mostraba correctamente la vista; sin embargo, ese error nos dio muchos dolores de cabeza hasta saber que así se solucionaba.

- Jiménez Hernández Diana

  Para crear el modelo conceptual se presentaron problemas para identificar algunos tipos de relaciones, el principal se debió a la relación entre inventario y artículo ya que inicialmente se modeló al artículo como débil ya que se requería que existiera en el inventario pero debido a que se utilizó un supertipo se optó por modificar y evitar la propagación de la llave primaria.

3. Recomendaciones para futuros proyectos:

   - Labastida Vázquez Fernando

     Si la base de datos crece mucho, asegúrate de que las consultas sean rápidas: A medida que tengas más información, puede hacer que el sistema se vuelva lento. Así que, asegúrate de revisar y mejorar las consultas para que el sistema siga funcionando rápido.

   - Badillo Aguilar Diego

     No te olvides de la seguridad: Si hay información importante en la base de datos, es crucial protegerla. Piensa en cosas como quién puede acceder a los datos y cómo mantenerlos seguros.

   - Jiménez Hernández Diana

     Si esperas que la base de datos siga creciendo, piensa en cómo hacerla más grande: Es posible que necesites pensar en cómo dividir la información o en otras técnicas para asegurarte de que el sistema siga funcionando bien a medida que crece.

   - Salgado Valdés Andrés

Recuerda mantener y actualizar la base de datos: No te olvides de hacer copias de seguridad, actualizar el software y revisar el rendimiento de vez en cuando.