UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERIA

División de Ingeniería Eléctrica

PROYECTO BASES DE DATOS

Profesor:
Fernando Arreola Franco

EQUIPO 3

Integrantes:
Castillo Bautista Samantha Lucia
Galindo Granados Abner Alejandro
Oliva Santiago Vania Nayeli
Sifuentes Cordero Carlos Emir

Grupo:
01

Fecha de entrega:
25 de mayo de 2025

Semestre 2025-2

ÍNDICE

1. INTRODUCCIÓN

En la actualidad, mantener los activos de una empresa únicamente en formato físico representa una desventaja significativa, ya que existe un alto riesgo de pérdida de información y los procesos de búsqueda y consulta suelen ser lentos e ineficientes. Ante este panorama, la transformación digital se ha vuelto una necesidad esencial para aquellas organizaciones que desean mantenerse competitivas, mejorar su eficiencia operativa y ofrecer un mejor servicio al cliente.

En este proyecto, se aborda un caso práctico que ejemplifica dicha necesidad: una mueblería que actualmente gestiona sus procesos de forma física y manual, lo cual genera múltiples limitaciones en la administración y acceso a la información.

El problema se divide en dos partes. La primera consiste en diseñar un modelo de base de datos que gestione eficientemente el inventario, proveedores, ventas y clientes de la empresa, integrando atributos clave como códigos de barras, precios, stock, datos fiscales y bancarios, así como registros detallados de ventas. Además, se requiere una gestión completa del personal, incluyendo jerarquías, roles, validaciones entre empleados y sucursales, generación de tickets de venta y reportes de inventario con alertas por bajo stock.

La segunda parte del proyecto contempla la creación de un dashboard de visualización, donde se representen indicadores de negocio relevantes. Para lo cual se desarrollado con ayuda de Power BI que esta vinculado a la base de datos y exporta los datos de esta para después generar los gráficos correspondientes que ayudaran a la toma de decisiones. Esta parte del sistema se considera crucial para facilitar la toma de decisiones basada en datos.

La implementación de esta solución implica la construcción de estructuras relacionales sólidas, el uso de herramientas de programación en base de datos como triggers y funciones, y el cumplimiento de reglas de negocio a través de restricciones y validaciones. Asimismo, se aplican técnicas de optimización mediante el uso de índices y consultas avanzadas, buscando siempre la eficiencia y precisión de la información.

Este proyecto no solo representa la aplicación práctica de los conocimientos adquiridos durante el curso, sino también una simulación de los procesos que intervienen en el desarrollo de una solución real, desde la especificación de requerimientos hasta la presentación de resultados. Con ello, se busca ofrecer una herramienta funcional, escalable y útil.

2. PLAN DE TRABAJO

El desarrollo del proyecto se llevó a cabo en varias etapas, siguiendo una planificación estructurada que permitió abordar de manera ordenada el análisis, diseño, implementación y documentación de la solución.

2.1. Fases del proyecto

- **Análisis de requerimientos.** Se revisó el docuento proporcionado, identificando los requerimientos funcionales y no funcionales de la base de datos.
- **Diseño de los modelos para la base de datos.** A partir del análisis de los requerimientos, se elaboraron modelos *entidad-relación* y *relacional*, asegurando una correcta definición de claves primarias y foráneas, y sobre todo del tipo de relación que había entre cada entidad.
- **Implementación de la base datos.** Para este paso, se creo el script *tablas.sql*, tomando como base el modelo relacional en PostgreSQL, incluyendo solamente las tablas, sus atributos, llaves foráneas y llaves primarias. Para este script se consideró DDL (*Data Definition Lenguage*).
- **Creación de funciones.** Se creo el script *funciones.sql* en el que se ingresaron los *triggers*, funciones que se ocuparon para completar las consideraciones, además de añadir el formato del folio de la venta mediante *sequence*, valor por *default* de la razón social del cliente, el stock del articulo, índice creado, validación del vendedor y cajero, ticket de venta y la jerarquía.
- **Carga de datos.** Se creo el script *datos.sql* en el que contiene las palabras reservadas de DML (*Data Manipulation Lenguage*) para agregar información a las tablas.
- **Dashboard de visualización.** Se desarrolló un panel de visualización de datos, con dos gráficas relevantes para la toma de decisiones dentro del negocio.
- **Documentación y presentación.** Se elaboró el documento en Latex y la presentación del proyecto.

2.2. Organización del equipo

- Samantha Castillo. Desarrollo inicial del modelo entidad-relación y relacional con la colaboración de mi compañero Abner Galindo para la identificación de las relaciones. Apoyo en la corrección de sintaxis de triggers y funciones ademas de la creación de indices para tener un mejor control de la información de los empleados que trabajan en las sucursales, entre otros.

Desarrollo de los dashboard mediante la aplicación de Power BI para obtener las gráficas que ejemplificarían de manera mas visual los datos obtenidos para el proyecto. Colaboración en este documento y la presentación correspondiente.

- Abner Galindo. Colaborar en la parte de la creación de tablas y principalmente en la creación de constrains en las llaves foráneas. Análsis y desarrollo de la lógica de las funciones para implementar las restricciones solicitadas en el documentos y creación de triggers para la parte de ventas. Colaboración en la parte de la creación de vistas. Colaboración dentro de este documento y la presentación.

- Vania Oliva. Colaborar en la parte de la realización del modelo Entidad-Relación, generar junto con mi compañero Carlos Sifuentes el DML correspondiente a la base de datos, analizar las restricciones de la base de datos, participar en el desarrollo de este documento y la presentación.

- Carlos Sifuentes. Analizar la transformación del modelo Entidad-Relación al modelo Relacional y participar en su elaboración, modificar partes del DDL conforme iba analizando más el problema buscando soluciones para que el diseño fuera el más adecuado y consistente con la información, participar junto con Vania en la parte del ingreso de datos de todas las tablas, participar en la elaboración de este documento y en la realización de la presentación.

3. DISEÑO

3.1. Análisis de requerimientos

Dado el requeriminto se obtienen los siguientes datos en los que nos basaremos para las siguientes fases de diseño.

Para artículo:

- Atributos: código de barras, nombre, precio de venta, precio de compra, stock y fotografía.
- Relaciones: cada artículo pertenece solo a una categoría, un artículo puede ser surtido por más de un proveedor.

Para proveedor:

- Atributos: RFC, razón social, dirección, teléfono y cuenta para pago
- Relaciones: un proveedor puede surtir muchos artículos.

Para venta:

- Atributos: folio, fecha, monto total, monto por artículo, cantidad total de artículos, cantidad por artículo.
- Relaciones: una venta puede incluir varios artículos, cada venta debe registrar quién la concretó y quién la cobró.

Para cliente:

- Atributos: RFC, nombre, razón social (por defecto es el nombre completo), dirección, correo electrónico y teléfono.
- Relaciones: no es obligatorio registrar al cliente para realizar una venta, pero la venta sí debe quedar registrada

Para empleado:

- Atributos: nombre, apellido paterno y materno, CURP, RFC, dirección, teléfonos, correo electrónico, fecha de ingreso y tipo de empleado (solo uno: cajero, vendedor, administrativo, seguridad o limpieza).
- Relaciones: cada empleado pertenece a una única sucursal, un empleado puede ser vendedor o cajero en una venta.

Para sucursal:

- Atributos: ubicación, teléfono y año de fundación.
- Relaciones: una sucursal puede tener múltiples empleados, cada venta debe involucrar empleados de la misma sucursal.

3.2. Modelo conceptual

Después de una análisis riguroso de los requerimientos planteados, realizamos nuestro modelo conceptual analizando aquellas entidades necesarias para poder realizar nuestra base de datos, con sus atributos correspondientes, además identificando aquellos atributos que nos permitirán identificar los datos de manera única, es decir las claves principales, también atributos que sean únicos (claves candidatas), atributos que puedan contener más de un valor (atributos multivaluados), atributos que puedan contener valores nulos (atributos opcionales) y que se puedan subdividir (atributos compuestos).

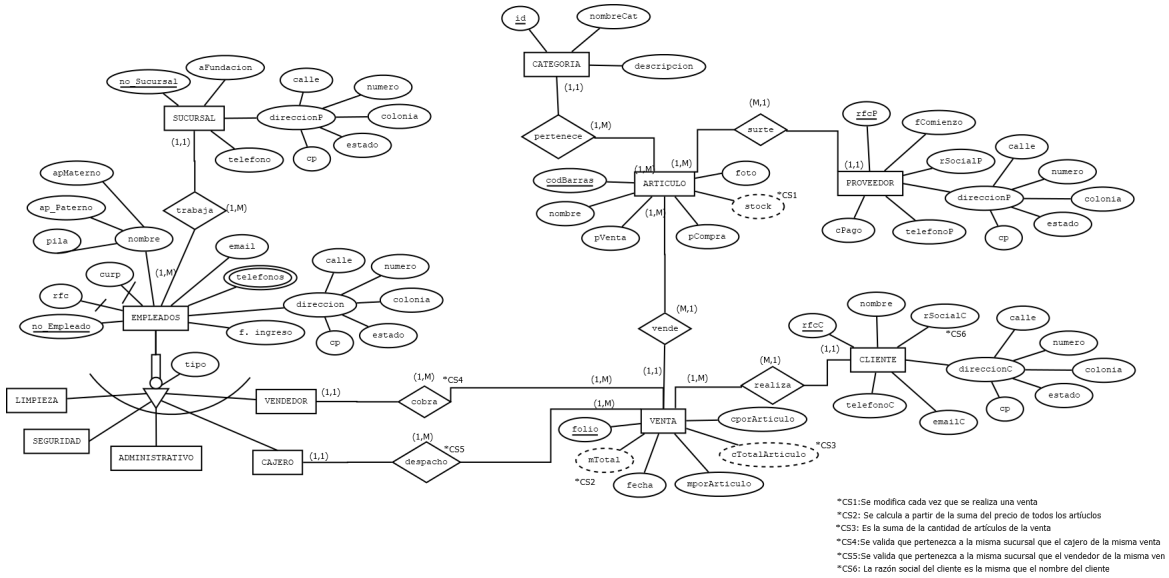

Figura 1: Modelo entidad relación propuesto

3.3. Modelo lógico

El modelo relacional se construyó a partir del modelo ER previamente elaborado, aplicando las reglas de transformación convencionales. Cada entidad se representó como una tabla, incluyendo sus atributos y definiendo la clave primaria correspondiente. En los casos en los que existían relaciones uno a muchos, se agregó una clave foránea en la entidad del lado "muchos". Para las

relaciones muchos a muchos, se crearon tablas intermedias con claves foráneas que referencian a ambas entidades participantes.

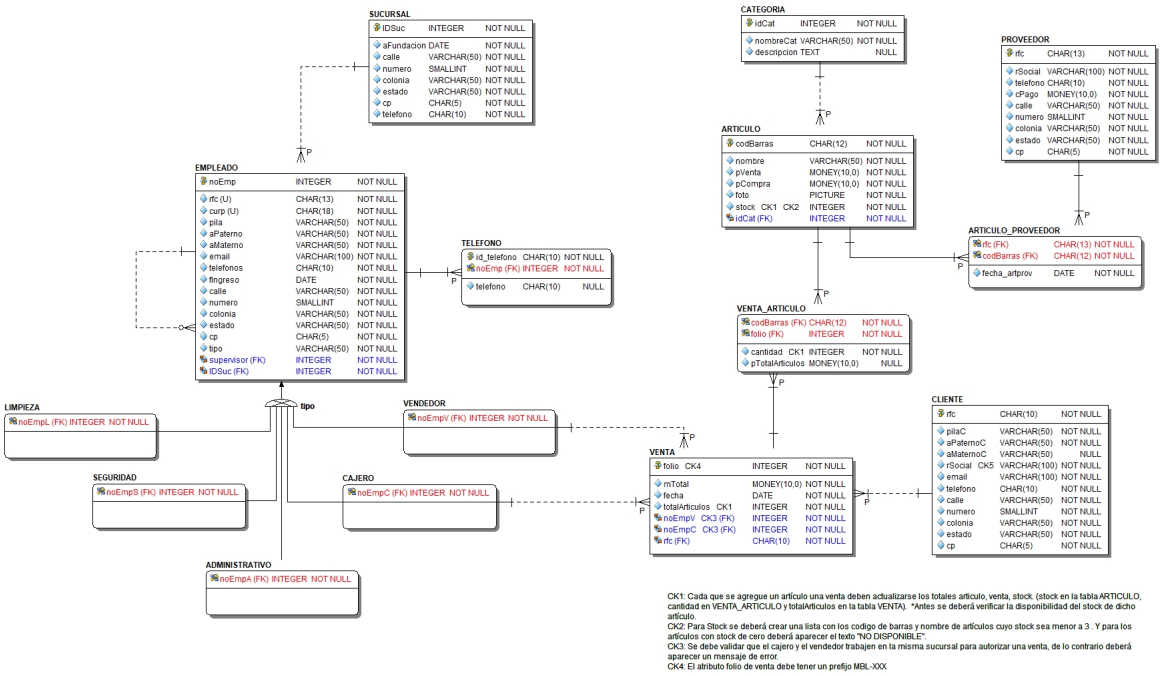

Figura 2: Modelo Relacional propuesto

3.4. Modelo físico

Para el modelo físico de la base de datos se creará en PostgreSQL en la base datos *proyectoBD* que fue creada para uso exclusivo del proyecto. Se crearon las tablas descritas anteriormente con su respectiva PK, atributos y sus FK's, las cuales son mostradas a continuación, cabe aclarar que sus consideraciones semánticas aún no son implementadas.

```
CREATE TABLE SUCURSAL(
    ID_SUCURSAL integer primary key,
    AÑO_FUNDACION char(4) not null,
    NUMERO smallint not null,
    CALLE varchar(50) not null,
    COLONIA varchar(50) not null,
    ESTADO varchar (50) not null,
    CODIGO_POSTAL char(5) not null,
    TELEFONO char(10) not null
);
```

Figura 3: Tabla Sucursal

```sql
CREATE TABLE CLIENTE(
    RFC_CLIENTE varchar(13) primary key,
    NOMBRE varchar(50) not null,
    APELLIDO_PATERNO varchar(50) not null,
    APELLIDO_MATERNO varchar(50) null,
    RAZON_SOCIAL varchar (100) not null,
    EMAIL varchar (100) not null,
    TELEFONO char (10) not null,
    CALLE varchar (50) not null,
    NUMERO smallint not null,
    COLONIA varchar(50) not null,
    ESTADO varchar (50) not null,
    CODIGO_POSTAL char(5) not null
);
```

Figura 4: Tabla Cliente

```sql
CREATE TABLE CATEGORIA(
    ID_CATEGORIA integer primary key,
    NOMBRE_CATEGORIA varchar(50) not null,
    DESCRIPCION text null
);
```

Figura 5: Tabla Categoría

```sql
CREATE TABLE PROVEEDOR(
    RFC_PROVEEDOR varchar(12) primary key,
    FECHA_INICIO date not null,
    RAZON_SOCIAL varchar(100) not null,
    TELEFONO char(10) not null,
    CUENTA_PAGO varchar (18) not null,
    CALLE varchar (50) not null,
    NUMERO smallint not null,
    COLONIA varchar(50) not null,
    ESTADO varchar (50) not null,
    CODIGO_POSTAL char(5) not null
);
```

Figura 6: Tabla Proveedor

```sql
CREATE TABLE ARTICULO(
    CODIGO_BARRAS varchar(13) primary key,
    NOMBRE varchar(50) not null,
    PRECIO_VENTA money not null,
    PRECIO_COMPRA money not null,
    FOTO text not null,
    STOCK integer not null,
    ID_CATEGORIA integer not null,
    CONSTRAINT ARTICULO_CATEGORIA_FK FOREIGN KEY (ID_CATEGORIA) REFERENCES CATEGORIA(ID_CATEGORIA)
    ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Figura 7: Tabla Artículo

```sql
CREATE TABLE EMPLEADO(
    NUMERO_EMPLEADO integer primary key,
    RFC_EMPLEADO varchar(13) not null UNIQUE,
    CURP_EMPLEADO varchar(18) not null UNIQUE,
    NOMBRE varchar(50) not null,
    APELLIDO_PATERNO varchar(50) not null,
    APELLIDO_MATERNO varchar(50) null,
    EMAIL varchar(100) not null,
    FECHA_INGRESO date not null,
    CALLE varchar (50) not null,
    NUMERO smallint not null,
    COLONIA varchar(50) not null,
    ESTADO varchar (50) not null,
    CODIGO_POSTAL char(5) not null,
    TIPO char (1) not null,
    ID_SUCURSAL integer not null,
    SUPERVISOR integer null,
    CONSTRAINT ID_SUCURSAL_FK FOREIGN KEY (ID_SUCURSAL) REFERENCES SUCURSAL(ID_SUCURSAL)
    ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT SUPERVISOR_FK FOREIGN KEY (SUPERVISOR) REFERENCES EMPLEADO(NUMERO_EMPLEADO)
    ON DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT ck_tipo_empleado CHECK(tipo IN ('A','C','L','S','V'))
);
```

Figura 8: Tabla Empleado

```sql
CREATE TABLE ADMINISTRATIVO(
    NUMERO_EMPLEADO_ADM integer not null,
    CONSTRAINT EMPLEADO_ADM_FK FOREIGN KEY (NUMERO_EMPLEADO_ADM) REFERENCES EMPLEADO(NUMERO_EMPLEADO)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT ADMIN_PK PRIMARY KEY (NUMERO_EMPLEADO_ADM)
);

CREATE TABLE LIMPIEZA(
    NUMERO_EMPLEADO_LIM integer not null,
    CONSTRAINT LIMPIEZA_FK FOREIGN KEY (NUMERO_EMPLEADO_LIM )REFERENCES EMPLEADO(NUMERO_EMPLEADO)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT LIMPIEZA_PK PRIMARY KEY (NUMERO_EMPLEADO_LIM)
);

CREATE TABLE SEGURIDAD(
    NUMERO_EMPLEADO_SEG integer not null,
    CONSTRAINT SEGURIDAD_FK FOREIGN KEY (NUMERO_EMPLEADO_SEG) REFERENCES EMPLEADO(NUMERO_EMPLEADO)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT SEGURIDAD_PK PRIMARY KEY (NUMERO_EMPLEADO_SEG)
);

CREATE TABLE VENDEDOR(
    NUMERO_EMPLEADO_VEN integer not null,
    CONSTRAINT VENDEDOR_FK FOREIGN KEY (NUMERO_EMPLEADO_VEN) REFERENCES EMPLEADO(NUMERO_EMPLEADO)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT VENDEDOR_PK PRIMARY KEY (NUMERO_EMPLEADO_VEN)
);

CREATE TABLE CAJERO(
    NUMERO_EMPLEADO_CAJ integer not null,
    CONSTRAINT CAJERO_FK FOREIGN KEY (NUMERO_EMPLEADO_CAJ) REFERENCES EMPLEADO(NUMERO_EMPLEADO)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT CAJERO_PK PRIMARY KEY (NUMERO_EMPLEADO_CAJ)
);
```

Figura 9: Tabla de los subtipos de Empleado

```sql
CREATE TABLE TELEFONO(
    NUMERO_EMPLEADO integer not null,
    TELEFONO char(10) null,
    CONSTRAINT NUM_EMP_TEL_PK PRIMARY KEY (NUMERO_EMPLEADO,TELEFONO),
    CONSTRAINT TEL_EMPLEADO_FK FOREIGN KEY (NUMERO_EMPLEADO) REFERENCES EMPLEADO(NUMERO_EMPLEADO)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

Figura 10: Tabla Teléfono

```
CREATE TABLE ARTICULO_PROVEEDOR(
    RFC_PROVEEDOR varchar(13) not null,
    CODIGO_BARRAS varchar (13) not null,
    FECHA_COMIENZO_SURTIDO_ART date not null,
    CONSTRAINT RFC_PROVEEDOR_FK FOREIGN KEY (RFC_PROVEEDOR) REFERENCES PROVEEDOR(RFC_PROVEEDOR)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT CODIGO_BARRAS_FK FOREIGN KEY (CODIGO_BARRAS) REFERENCES ARTICULO(CODIGO_BARRAS)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT ART_PROV_PK PRIMARY KEY(RFC_PROVEEDOR,CODIGO_BARRAS)
);
```

Figura 11: Tabla ArticuloProveedor

```
CREATE TABLE VENTA(
    FOLIO varchar(10) primary key,
    MONTO_TOTAL money,
    FECHA_VENTA date not null,
    TOTAL_ARTICULOS integer,
    NUMERO_EMPLEADO_VEN integer not null,
    NUMERO_EMPLEADO_CAJ integer not null,
    RFC_CLIENTE varchar(13),
    FOLIO_FACTURACION varchar(12),
    CONSTRAINT V_VENDEDOR_FK FOREIGN KEY (NUMERO_EMPLEADO_VEN) REFERENCES VENDEDOR(NUMERO_EMPLEADO_VEN)
    ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT V_CAJERO_FK FOREIGN KEY (NUMERO_EMPLEADO_CAJ) REFERENCES CAJERO(NUMERO_EMPLEADO_CAJ)
    ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT folio_facturacion_unico UNIQUE (folio_facturacion),
);
```

Figura 12: Tabla Venta

```
CREATE TABLE VENTA_ARTICULO(
    CODIGO_BARRAS varchar(13) not null,
    FOLIO varchar (10) not null,
    CANTIDAD integer not null,
    PRECIO_TOTAL_ARTICULOS money null,
    CONSTRAINT VA_COD_BAR_FK FOREIGN KEY (CODIGO_BARRAS) REFERENCES ARTICULO(CODIGO_BARRAS)
    ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT VA_FOLIO FOREIGN KEY (FOLIO) REFERENCES VENTA(FOLIO)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT VA_PK PRIMARY KEY (CODIGO_BARRAS,FOLIO)
);
```

Figura 13: Tabla VentaArticulo

4. IMPLEMENTACIÓN

4.1. Secuencias

Folio de la venta. Para la creación del folio, primero se crea la secuencia *ventafolioseq* que empezará desde 1. Se modifica la columna FOLIO de la tabla VENTA, estableciendo un valor por defecto que se generará automáticamente cuando no se especifique un valor al insertar una nueva fila. Considerando que *nextval('ventafolioseq')* obtiene el siguiente número de la secuencia, *::TEXT* convierte el número a texto, *LPAD(..., 6, '0')* completa con ceros a la izquierda hasta que se tengan 6 dígitos y *'MBL-'* antepone el prefijo para el folio.

```
CREATE SEQUENCE venta_folio_seq START 1;

ALTER TABLE VENTA
ALTER COLUMN FOLIO SET DEFAULT 'MBL-' || LPAD(nextval('venta_folio_seq')::TEXT, 6, '0');
```

Figura 14: Implementación para el folio facturación

Folio para facturación. El valor por defecto para la facturación empieza con el prefijo *FAC-*, *random()* devuelve un número decimal aleatorio entre 0 y 1, *::TEXT* convierte el número a texto, *md5* calcula el hash MD5 de ese texto (cadena de 32 caracteres en hexadecimal), se extraen los primeros 8 caracteres y se concatena con el prefijo.

```
ALTER TABLE VENTA
ALTER COLUMN FOLIO_FACTURACION
SET DEFAULT 'FAC-' || substr(md5(random()::text), 1, 8);
```

Figura 15: Implementación para el folio de la venta con formato MBL-XXXXXX

4.2. Funciones y triggers

Razón social como nombre por default. Para colocar como default el nombre del cliente en razón social creamos una función que retorna un trigger. Esta función verifica si la razón social ingresada para es cliente es nula o tiene un espacio (es decir no se especificó una razón social), si es verdadero a la razón social se le asignará el nombre del cliente y concatenamos con || ' ' un espacio y después concatenamos el apellido paterno y después el apellido materno. En esta parte sabemos que el apellido materno puede ser nulo, entonces con *COALESCE* nos aseguramos de que en caso de que el apellido materno sea nulo se coloca un espacio en ese registro, en otro caso se coloca el apellido. Se termina la función y regresamos el valor que se editó. Esto lo hacemos mediante la variable new que almacena la fila que se insertó y en el return new nos indica que se debe utilizar

la fila modificada. Por último se crea el trigger sobre la tabla cliente con las especificaciones que se dispare antes de cada inserción y para cada fila.

```
CREATE FUNCTION razon_social_default() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.RAZON_SOCIAL IS NULL OR NEW.RAZON_SOCIAL = '' THEN
        NEW.RAZON_SOCIAL := NEW.NOMBRE || ' ' || NEW.APELLIDO_PATERNO ||
                                COALESCE(' ' ||NEW.APELLIDO_MATERNO, '');
    END IF;
    RETURN NEW;
END; $$
LANGUAGE PLPGSQL;
```

Figura 16: Función razón social default

```
CREATE TRIGGER razon_social_default
BEFORE INSERT ON CLIENTE
FOR EACH ROW
EXECUTE FUNCTION razon_social_default();
```

Figura 17: Trigger sobre tabla cliente

Validación de Vendedor y Cajero. Para esta restricción se creó una función que nos regresa un trigger. Se declaran dos variables sucursal_ven y sucursal_caj. Se harán dos consultas que se van a almacenar en las variables para el cajero y para el vendedor; se selecciona el id_sucursal de la tabla empleado en donde el numero del empleado coincida con el numero del empleado (cajero, vendedor) que se insertó en esa fila que fue el que atendió esa venta. Entonces en las variables declaradas se almacena la sucursal para la que trabajan. Después se hace la validación en dónde si la sucursal para la que trabaja el cajero es distinta a la sucursal del vendedor se mandará un mensaje indicándolo. En esta parte se utilizó *RAISE EXCEPTION* ya que nos ayuda a detener la ejecución de la función. En caso de que la condición no se cumpla deja que se realice la venta. Por último, se crea el trigger sobre la tabla venta con las especificaciones que se dispare antes de cada inserción y para cada fila.

```
CREATE FUNCTION validacion_empleados() RETURNS TRIGGER AS $$
DECLARE
    sucursal_ven int;
    sucursal_caj int;
BEGIN
    SELECT ID_SUCURSAL into sucursal_ven from empleado where numero_empleado = new.numero_empleado_ven;
    SELECT ID_SUCURSAL into sucursal_caj from empleado where numero_empleado = new.numero_empleado_caj;

    IF sucursal_ven is distinct from sucursal_caj then
    raise exception 'El cajero y vendedor no pertenecen a la misma sucursal, no se puede realizar la venta';
    end if;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;
```

Figura 18: Función validación de empleados

```
CREATE TRIGGER validacion_empleados BEFORE INSERT ON VENTA
FOR EACH ROW
EXECUTE FUNCTION validacion_empleados();
```

Figura 19: Trigger sobre tabla Venta

Calculo del precio por artículos. Para este cálculo se creó una función que nos regresa un trigger. Se declara una variable llamado precio_ unitario. Se hace una consulta en donde vamos a seleccionar el precio de venta de la tabla articulo en donde su código de barras coincida con el código de barras del artículo que se va a comprar, esta consulta se almacena en la variable que declaramos, ya que tenemos ese valor a la variable PRECIO_TOTAL_ARTICULOS se le va asignar el resultado de la multiplicación del precio unitario de cada artículo por la cantidad que se seleccionó y regresamos el nuevo valor que aparecerá en la tabla. Creamos el trigger sobre la tabla VENTA_ARTICULO con las especificaciones que se va a disparar antes de cada inserción para tener ya ese valor en cada articulo que se agregue, y por cada fila.

```
CREATE FUNCTION calculo_total_por_articulos() RETURNS TRIGGER AS $$
DECLARE
    precio_unitario money;
BEGIN
    SELECT PRECIO_VENTA INTO precio_unitario FROM ARTICULO WHERE CODIGO_BARRAS = NEW.CODIGO_BARRAS;
    NEW.PRECIO_TOTAL_ARTICULOS := precio_unitario * NEW.CANTIDAD;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;
```

Figura 20: Función del cálculo del precio por artículos

```
CREATE TRIGGER calculo_total_por_articulos BEFORE INSERT ON VENTA_ARTICULO
FOR EACH ROW
EXECUTE FUNCTION calculo_total_por_articulos();
```

Figura 21: Trigger sobre la tabla VENTA_ARTICULO

Cálculo del monto total. Para este calculo se creó una función que nos regresa un trigger. Se hace un update al atributo monto_total de la tabla venta, para el valor que va a tomar se utilizarán los cálculos que regresó la anterior función que se creó. Se hace una consulta a la tabla VENTA_ARTICULO donde seleccionamos la suma de la columna precio_total_articulos, pero la función de sum está dentro de la función *COALESECE* para que en caso de que se encuentre un valor de null se cambie por un 0, este 0 lo vamos a castear con *0::money* ya que 0 es de tipo entero y la columna tiene como tipo de dato money. Así se sumará un 0 que no afectará al cálculos. Se hará esta consulta en dónde el folio coincida, es decir, en dónde se trate de la misma venta. Fuera de esa subconsulta de igual forma el update se hace en dónde los folios coincidan, actualiza solo la venta correspondiente. Se crea el trigger sobre la tabla VENTA_ARTICULO con las especificaciones que se dispare después de cada inserción para cada fila.

```
CREATE FUNCTION calculo_monto_total() RETURNS TRIGGER AS $$
BEGIN
    UPDATE VENTA SET MONTO_TOTAL =
    (SELECT COALESCE(SUM(PRECIO_TOTAL_ARTICULOS),0::money) FROM VENTA_ARTICULO WHERE FOLIO = NEW.FOLIO)
    WHERE FOLIO = NEW.FOLIO;
    RETURN NULL;
END; $$
LANGUAGE plpgsql;
```

Figura 22: Función del cálculo del monto total

```
CREATE TRIGGER calculo_monto_total AFTER INSERT ON VENTA_ARTICULO
FOR EACH ROW
EXECUTE FUNCTION calculo_monto_total();
```

Figura 23: Trigger sobre la tabla VENTA_ARTICULO

Suma del total de artículos. Para este cálculo se creó una función que nos regresa un trigger. Esta función es parecida a la del calculo del monto total ya que se hace un update al atributo TOTAL_ARTICULOS de la tabla Venta. Para asignar el valor se hace una consulta a la tabla VENTA_ARTICULO donde seleccionamos la suma de la columna cantidad, pero la función de sum está dentro de la función *COALESECE* para que en caso de que se encuentre un valor de null se cambie por un 0, así se sumara un 0 que no afectará el cálculo. Se hará esta consulta en dónde el folio coincida, es decir, en dónde se trate de la misma venta. Fuera de esa subconsulta de igual forma el update se hace en dónde los folios coincidan, actualiza solo la venta correspondiente. Se crea el trigger sobre la tabla VENTA_ARTICULO con las especificaciones que se dispare después de cada inserción para cada fila.

```
CREATE FUNCTION suma_total_articulos() RETURNS TRIGGER AS $$
BEGIN
    UPDATE VENTA SET TOTAL_ARTICULOS =
    (SELECT COALESCE(SUM(CANTIDAD),0) FROM VENTA_ARTICULO WHERE FOLIO = NEW.FOLIO)
    WHERE FOLIO = NEW.FOLIO;
    RETURN NULL;
END; $$
LANGUAGE plpgsql;
```

Figura 24: Función de la suma del total de artículos

```
CREATE TRIGGER suma_total_articulos AFTER INSERT ON VENTA_ARTICULO
FOR EACH ROW
EXECUTE FUNCTION suma_total_articulos();
```

Figura 25: Trigger sobre la tabla VENTA_ARTICULO

Validación de disponibilidad de artículos. Para esta validación se utilizó una función que nos regresa un trigger. Se declara una variable llamada stock_actual de tipo entero. Primero se hace una consulta al stock en la tabla artículo que tiene el artículo que se va a comprar y se guarda dentro de la variable declarada, con este valor haremos dos validaciones. La primera es que si el stock actual es igual a 0 manda un mensaje indicando que el artículo no cuenta con stock

y regresa null, esto hace que el artículo no pueda ser agregado a la venta que se está realizando. Después se hace la segunda validación en dónde si el stock es menor a la cantidad solicitada por el cliente manda un mensaje diciendo que no hay suficiente stock para la demanda y muestra el stock disponible, de igual forma regresa null para no permitir que se ingrese ese artículo a la venta. En caso de no cumplir ninguna validación permite el ingreso del artículo de forma correcta. Se crea el trigger en la tabla ARTICULO_VENTA con las especificaciones que se dispare antes de cada inserción y para cada fila.

```
CREATE FUNCTION disponibilidad_articulo() RETURNS TRIGGER AS $$
DECLARE
stock_actual int;
BEGIN
    SELECT STOCK INTO stock_actual FROM ARTICULO WHERE CODIGO_BARRAS = NEW.CODIGO_BARRAS;
    IF stock_actual = 0
    THEN
    RAISE NOTICE 'El artículo con código % que desea agregar no está disponible',NEW.CODIGO_BARRAS;
    RETURN NULL;
    END IF;

    IF stock_actual < NEW.CANTIDAD
    THEN
    RAISE NOTICE 'El artículo con código % no cuenta con stock suficiente, stock disponible: %',NEW.CODIGO_BARRAS,stock_actual;
    RETURN NULL;
    END IF;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;
```

Figura 26: Función que valida la disponibilidad del artículo

```
CREATE TRIGGER disponibilidad_articulo BEFORE INSERT ON VENTA_ARTICULO
FOR EACH ROW
EXECUTE FUNCTION disponibilidad_articulo();
```

Figura 27: Trigger sobre tabla VENTA_ARTICULO

Resta del stock de artículos comprados. Para restar el stock se creó una función que nos regresa un trigger. Se va a modificar la columna Stock de la tabla artículo, al stock total del artículo se le va a restar la cantidad que solicitó el cliente del artículo que se va a comprar y va a regresar esa fila ya modificada. Se crea el trigger en la tabla VENTA_ARTICULO con las especificaciones que se dispare después de cada inserción y para cada fila, es decir cuando se agregue los artículos a la venta se resta el stock.

```
CREATE FUNCTION restar_stock() RETURNS TRIGGER AS $$
BEGIN
    UPDATE Articulo SET Stock = stock - new.cantidad WHERE CODIGO_BARRAS = NEW.CODIGO_BARRAS;
    RETURN NEW;
END; $$
LANGUAGE plpgsql;
```

Figura 28: Función que resta el stock de artículos comprados

```
CREATE TRIGGER restar_stock AFTER INSERT ON VENTA_ARTICULO
FOR EACH ROW
EXECUTE FUNCTION restar_stock();
```

Figura 29: Trigger definido sobre tabla VENTA_ARTICULO

Jerarquía organizacional de empleados. Se crea la función *jerarquia-empleado*, toma como parámetro el *numeroEmpleado*, devuelve una tabla virtual con columnas como número de empleado, nombre, apellidos, supervisor, sucursal y nivel. Con la sentencia *WITH RECURSIVE* se crea una consulta recursiva en la que encuentra al empleado que coincide con el parámetro de entrada, le asigna el nivel = 1 en la jerarquía. Se busca al supervisor del empleado anterior, se une *sup supervisor* y *j nivel anterior*, *j.supervisor* es el número de empleado del siguiente jefe en la jerarquía, asegurando que estén en la misma sucursal. El proceso lo repite hasta que no se encuentren más supervisores.

Para ejecutar y visualizar el resultado, se ingresa la sentencia SELECT * FROM JERARQUIAEMPLEADO(*numeroEmpleado*)

```
CREATE OR REPLACE FUNCTION jerarquia_empleado(emp_id INT)
RETURNS TABLE (numero_empleado INT,nombre VARCHAR(50),apellido_paterno VARCHAR(50),apellido_materno VARCHAR(50),
  supervisor INT,id_sucursal INT,nivel INT)
AS $$
BEGIN RETURN QUERY
WITH RECURSIVE jerarquia AS ( SELECT e.numero_empleado,e.nombre,e.apellido_paterno,e.apellido_materno,
      e.supervisor,e.id_sucursal,
      1 AS nivel
    FROM empleado e
    WHERE e.numero_empleado = emp_id
    UNION ALL
    SELECT sup.numero_empleado,sup.nombre,sup.apellido_paterno,sup.apellido_materno,
      sup.supervisor,sup.id_sucursal,
      j.nivel + 1
    FROM empleado sup
    INNER JOIN jerarquia j ON sup.numero_empleado = j.supervisor
    WHERE sup.id_sucursal = j.id_sucursal
  )
  SELECT * FROM jerarquia;
END;
$$ LANGUAGE plpgsql;
```

Figura 30: Función para jerarquía de empleado

4.3. Vistas

Vista para el stock. Para esta parte se creó una vista que va a mostrar el código de barras, el nombre del artículo y su stock y el estado del artículo. Dentro del estado se tendrán 2 casos. El primero si el stock es igual a 0 aparecerá como No disponible, el segundo si es menor a 3 aparecerá como stock bajo. Se seleccionará de la tabla artículo para los artículos cuyo stock sea menor que 3. Y se coloca la consulta a la vista.

```
CREATE VIEW restriccion_stock as
SELECT CODIGO_BARRAS, NOMBRE, STOCK,
CASE WHEN STOCK = 0 THEN 'NO DISPONIBLE'
ELSE 'STOCK BAJO'
END AS ESTADO
FROM ARTICULO WHERE STOCK < 3;
SELECT * FROM restriccion_stock;
```

Figura 31: Vista para los artículos con bajo stock

Vista para el ticket. Se crea la vista *vistaTicketVenta* la cual incluirá información de las

tablas venta, ventaArticulo y articulo, además ocupa *COALESCE* muestra si la venta tiene un folio de facturación (factura generada) y si es NULL (no se ha facturado), muestra la cadena 'NO FACTURADO'. Se crean 2 Join's para relacionar la tabla venta con la ventaArticulo por medio del folio y la tabla ventaArticulo con la tabla venta por medio del código de barras.

```sql
CREATE OR REPLACE VIEW vista_ticket_venta AS
SELECT
    v.folio AS folio_venta,
    v.fecha_venta,
    v.rfc_cliente,
    va.codigo_barras,
    a.nombre AS nombre_articulo,
    va.cantidad,
    a.precio_compra ,
    va.precio_total_articulos,
    v.monto_total AS total_venta,
    COALESCE(v.folio_facturacion, 'NO FACTURADO') AS folio_facturacion
FROM venta v
JOIN venta_articulo va ON v.folio = va.folio
JOIN articulo a ON va.codigo_barras = a.codigo_barras;
```

Figura 32: Vista para el ticket

4.4. Índices

Índice en sucursal. Este índice crea una estructura de búsqueda eficiente sobre la columna *idSucursal* en la tabla empleado. La columna *idSucursal* no es clave primaria en empleado, pero: es una clave foránea que apunta a la tabla sucursal, es el principal criterio de agrupación en empleados. El índice ayuda en la función de *validacion-empleados*, esta función se ejecuta cada vez que se hace una venta, por lo que este índice mejora su rendimiento, también ayuda en la función *jerarquia-empleado*. El índice es de tipo NONCLUSTERED.

```sql
CREATE INDEX CONCURRENTLY idx_empleado_sucursal ON empleado(id_sucursal);
```

Figura 33: IDX en tabla Sucursal

5. APLICACIÓN

Esto se desarrollo con Power BI y la vinculación a nuestra base de datos en PostgreSQL, esto nos sirve para que a nuestro cliente dueño de la mueblería pueda tomar decisiones en base a los datos que ha obtenido. Para llevar esto acabo se hizo una vinculación entre estos, en Power Bi podemos actualizar los datos que ingresamos a nuestra base de datos de una manera sencilla asi quien consulte los modelos podra tomar una desision acertada sin la necesidad de leer tablas con muchos datos que pueden causar confusiones. La gráficas que se desarrollaron fueron:

- Visualizar las ventas que tenemos mensualmente al año en cada sucursal: Esto nos ayudaria a saber en que sucursales estaos teniendo una mayor afluencia y en cuales menos para que en el caso de tener pocas ventas realizar alguna campaña de publicidad o crear promociones par atraer a mas clientes.

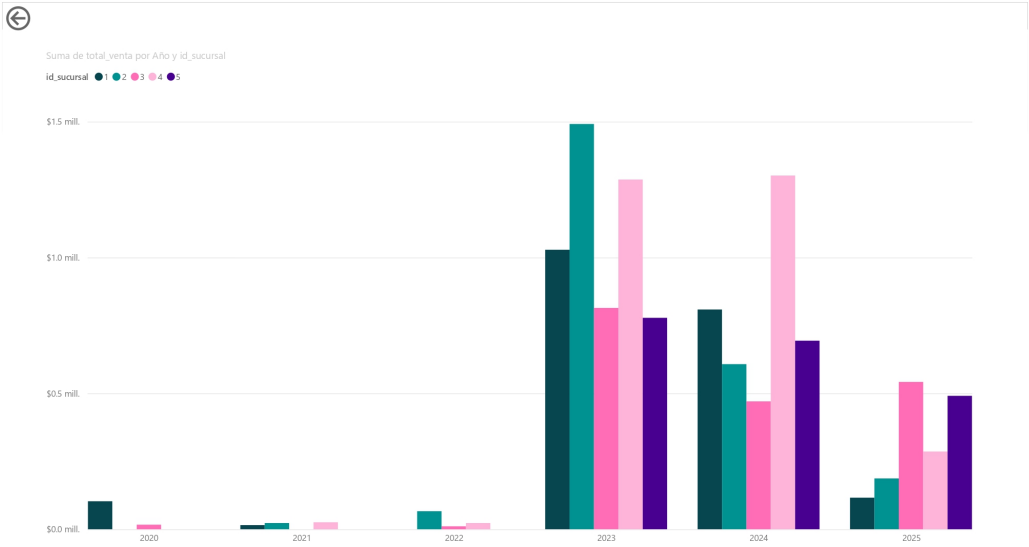

Figura 34: Gráfica ventas anuales

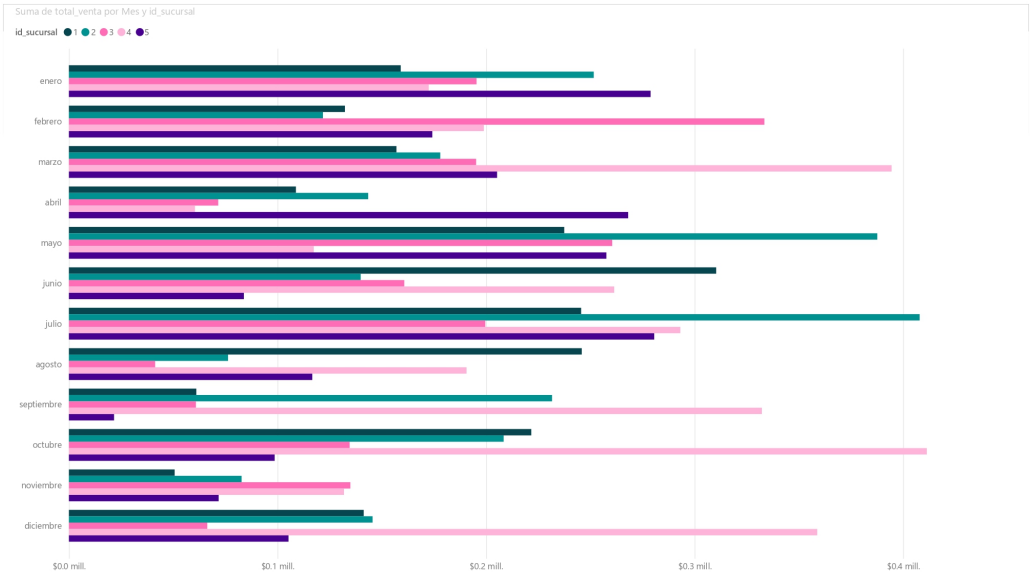

Figura 35: Gráfica ventas mensuales

- Comparativa de precio de ventas y compra en relación con el stock, así podemos visualizar cuales son nuestras áreas de oportunidad en los productos que se venden mas dependiendo del stock disponible.

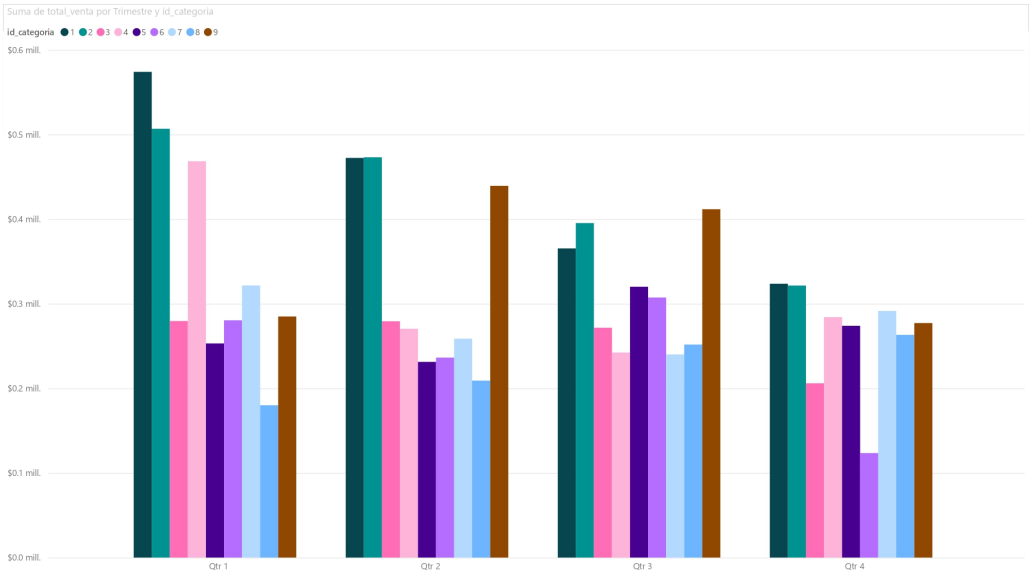

Figura 36: Gráfica compartiva precios venta-compra

- Producto mas vendido de manera trimestral que nos ayuda a visualizar en que categoria se puede incluir una mayor cantidad de productos ya que estos estan siendo mas solicitados por

nuestros clientes.

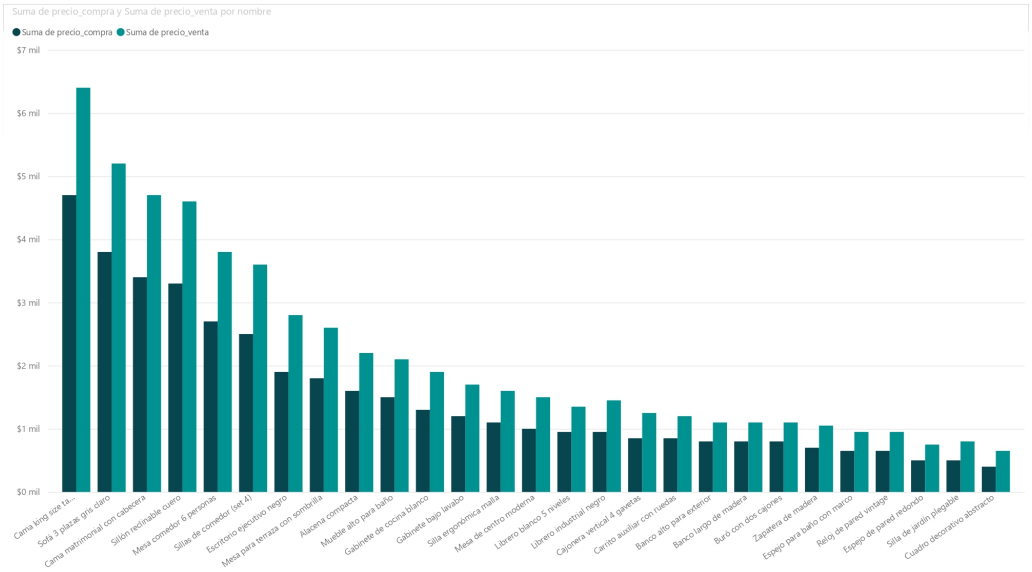

Figura 37: Gráfica productos mas vendidos trimestralmente

Mediante esta herramienta podemos obtener multiples representaciones visuales claras, donde es posible identificar rápidamente patrones, tendencias y comparaciones que respaldan las decisiones estratégicas propuestas en el proyecto. Destacando de forma efectiva los ingresos generados por cada sucursal en distintos periodos, lo que resulta esencial para evaluar su rentabilidad, crecimiento y áreas de oportunidad. Además, mejoran la comunicación del análisis con los diferentes públicos involucrados, especialmente con tomadores de decisiones que requieren información concisa y visualmente accesible.

6. CONCLUSIONES

Samantha Castillo. Considero que ha sido un proyecto retador porque al ser una materia completamente nueva muchas cosas las tuvimos que investigar como se podian realizar de manera mas eficiente, en varias ocasiones se consulto con el profesor para obtener una opinión con mas experiencia. Para el desarrollo del proyecto se debió de repartir el trabajo una desventaja es que la mayoría de las herramientas que ocupamos para desarropar el modelo entidad-relacion y relacional es que no era colaborativas entonces de manera constante se debía mandar los archivos para que todos los pudiéramos revisar y aprobar. Para el desarrollo de base de datos tuvimos problemas para conectarnos al servidor de la facultad por lo que se tomo la decisión de que cada quien lo hiciera en su dispositivo personal, esto presento que al momento del desarrollo debía de existir una buena comunicación para mantenernos a al tanto de los cambios realizados. Para el desarrollo de este aspecto al tener un buen modelo desde un inicio fue fácil apegarnos a el y desarrollarlo aunque se tuvieron que realizar algunas modificaciones en el ddl como la imagen ya que al investigar como agregarla con el tipo de dato 'bytea' vimos que era algo complejo por lo que se opto por una solución con una url de manera mas practica esto debido a las limitantes de tiempo. Otro punto que me pareció muy interesante de trabajar fueron los dashboards ya que no conocía esa manera de vincular ambas aplicaciones. Me pareció un proyecto interesante y enriquecedor que me da una perspectiva de como podría ser el trabajo en un futuro si decido dedicarme a esta área.

Abner Galindo. Ha sido un proyecto que nos puso muchos retos y en general nos ha generado trabajo, desde el la organización del equipo el como se van a dividir las tareas hasta la implementación y el desarrollo del mismo. En la parte de los modelos tanto entidad-relación y relacional lo manejamos de buena manera ya que los construimos de forma muy cercana al modelo final que obtuvimos, durante el desarrollo del ddl se tuvieron que modificar ciertas cosas que veíamos que eran mas optimas para el desarrollo de la base de datos. En la parte de ddl no se tuvo de igual forma tantos problemas ya que al haber tenido buenos modelos previos las tablas tenían una buena estructura, sin embargo conforme se fue avanzando se fuerón cambiando algunos tipos de datos, se fueron añadiendo constraints a las tablas, atributos, etc. Por último en la programación, consultas e inserción de datos fue que se tuvieron algunos problemas ya que en la inserción de datos llegabamos a tener distintos datos cada quién y no coincidian, en las funciones eran complicado poder modificarla ya que al no trabajar todos en un mismo servidor teníamos que compartir el script para poder tener la misma. Pienso que lo más complicado fue que no pudimos trabajar dentro del servidor de la facultad ya que a veces no estaba abierto, debido a eso teníamos que compartirnos los script con frecuencia para tratar todos de ir a la par con los datos, códigos, etc. Logramos organizarnos bien, logramos llegar a acuerdos como equipo y sobre todo trabajar de buena forma, creo ha sido un proyeto bastante bueno, algo bastante parecido a lo que será en

un mundo laboral y nos deja una experiencia y enseñanzas hacia futuros proyectos.

Vania Oliva. La elaboración de un proyecto de esta magnitud fue un gran reto, ya que todo comenzó a partir de un solo párrafo. Ese es precisamente uno de los aspectos más sorprendentes del proyecto: que a partir de poca información, es posible diseñar múltiples soluciones, tomar decisiones estructurales y construir un sistema funcional. Durante el desarrollo, fuimos descubriendo que las cosas pueden sobre la marcha. Por ejemplo, modificamos tipos de datos, añadimos atributos no ingresados en el modelo ER. Estas decisiones fueron tomadas por necesidades técnicas como de mejorar la utilidad del sistema. El trabajo con los modelos (entidad-relación y relacional) fue uno los elementos principales del proyecto. Tener una base nos permitió avanzar en la definición de las tablas y restricciones durante la fase DDL. Donde más dificultades tuvimos fue en la programación de funciones y la inserción de datos. Debido a que no teníamos un servidor compartido generó realizar gran cantidad de scripts. pesar de estos obstáculos, logramos organizarnos como equipo, distribuir el trabajo con eficacia, resolver conflictos técnicos y avanzar de manera colaborativa. Este proyecto no solo fortaleció nuestras habilidades en el diseño e implementación de bases de datos, sino que también nos brindó una experiencia muy cercana a los retos reales que se presentan en el ámbito laboral.

Carlos Sifuentes. Desde mi punto de vista este fue un proyecto bastante interesante con el que pudimos aplicar todo lo visto a lo largo del semestre, considero que esta es la naturaleza de la dificultad de su elaboración, partiendo desde la elaboración de su modelo conceptual, analizando los requerimientos, siempre dándonos cuenta que podríamos encontrar una mejor solución. De manera particular considero que el mayor reto de este proyecto esta relacionado con la venta, dada la cantidad de información que necesitabamos, además de automatizarlas con el uso de triggers, en un punto nos encontramos con una paradoja bastante curiosa por lo que tuvimos que poner atributos como el monto total y el total de articulos como nulos, de esta manera podíamos generar el folio de la venta para después usar este en el desglose de la venta con artículo. Por otra parte el proyecto tuvo otra enorme complejidad que fue la elaboración de los disparadores dada la nula información que teníamos en un principio sobre estos mecanismos que serían fundamentales en el desarrollo del proyecto, sin embargo considero que este reto lo sorteamos de manera adecuada. Personalmente me gustó bastante realizar este proyecto y ver la parte teórica de una manera mucho más práctica y enfocada en algo, considero que la parte más tediosa fue el ingreso de las ventas, ya que fueron bastantes y a cada una le corresponden varios ingresos en la tabla venta con articulo. Finalmente considero que la información de los dashboards es bastante útil dado el análisis que se puede sacar de estos datos.