1. Introducción

Este proyecto busca diseñar y crear una base de datos para un negocio que vende cotillón. Es importante tener un sistema organizado para manejar todos los productos variados, controlar el inventario, registrar las ventas y administrar la información de los clientes y proveedores.

La idea principal es desarrollar una base de datos que permita manejar todo lo relacionado con las operaciones del negocio de cotillón: desde la administración de productos y el control del stock, hasta el registro de clientes, ventas y proveedores. También se busca que esta base pueda generar reportes útiles para analizar el negocio y tomar mejores decisiones, considerando aspectos contables y logísticos vinculados al inventario y las ventas.

2. Situación problemática

Hoy en día, muchos negocios de cotillón trabajan con información manual o con sistemas poco conectados entre sí, lo que provoca problemas como falta de control sobre el stock, errores al facturar, dificultad para conocer el historial de compras de los clientes y poca claridad sobre cómo está funcionando el negocio. Esto complica la operación diaria y limita la planificación de compras y ventas.

Con una base de datos bien diseñada, se puede solucionar todo esto, ofreciendo un sistema centralizado, confiable y fácil de usar que ayude a llevar un mejor control, reducir errores y tener información precisa y actualizada en cualquier momento.

3. Modelo de negocio

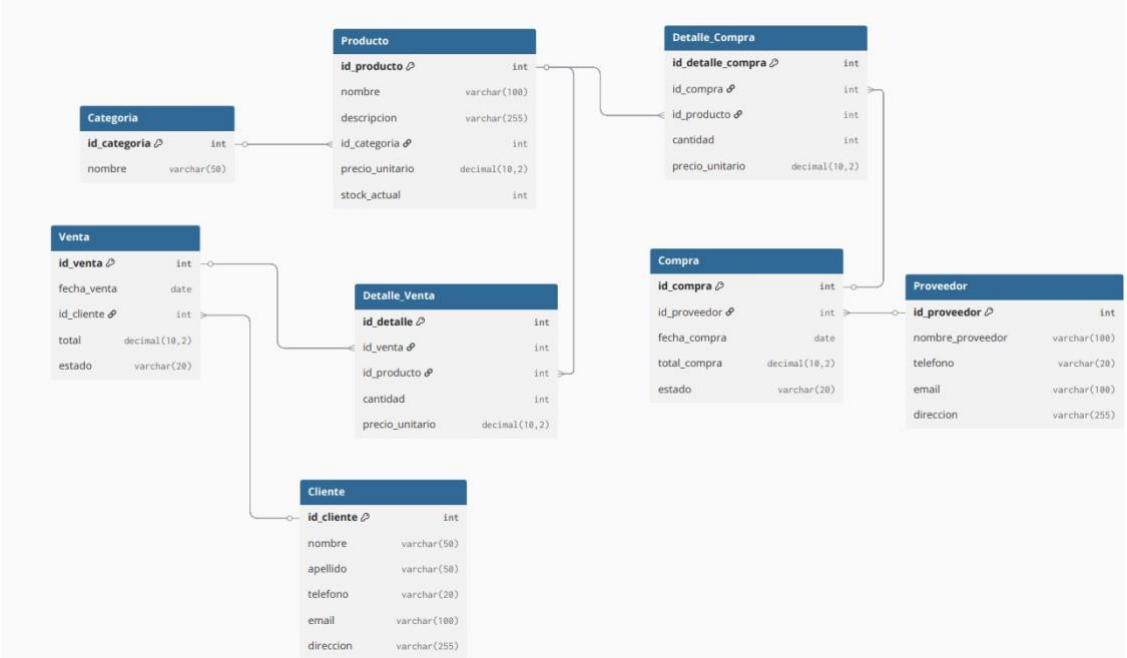
El negocio de cotillón suele ser una empresa pequeña o mediana que vende productos para fiestas y celebraciones, como globos, decoraciones, confites y artículos temáticos. Compra sus productos a distintos proveedores y atiende tanto a clientes particulares como a organizadores de eventos.

El foco principal está en manejar el inventario de manera eficiente y brindar una buena atención a los clientes, ofreciendo una amplia variedad de productos con precios competitivos. Por eso, la base de datos deberá registrar información clave como las categorías de productos, precios, stock disponible, datos de los clientes y los movimientos de venta.

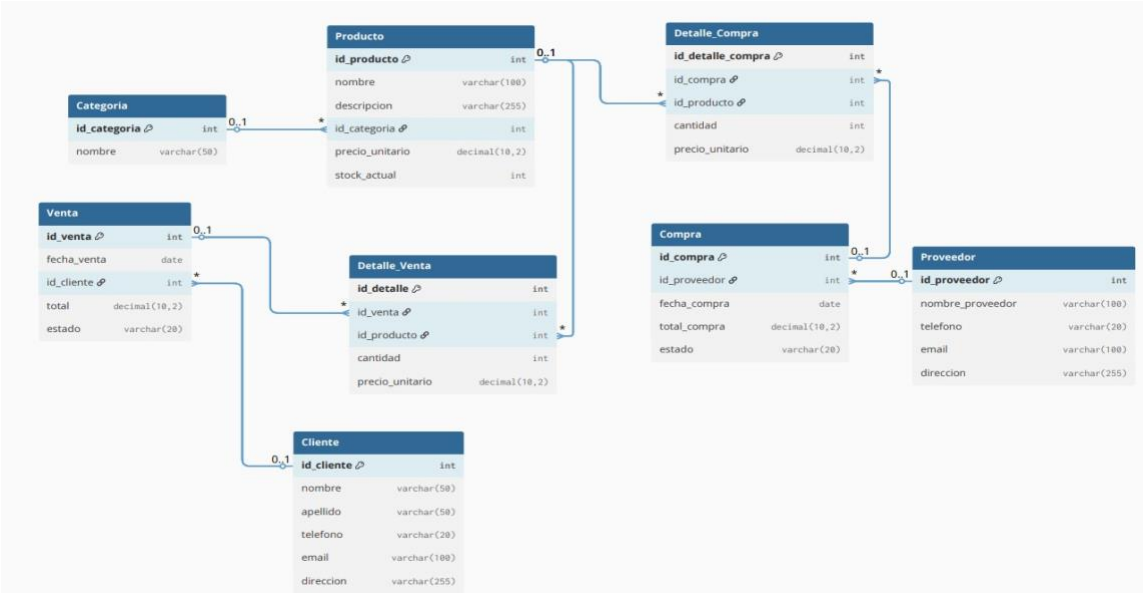
El Diagrama Entidad-Relación (DER) representa la estructura de la base de datos diseñada para mi proyecto. En él se visualizan todas las

entidades involucradas, como Producto, Cliente, Proveedor, Venta y Compra, junto con sus relaciones y cardinalidades, que reflejan cómo interactúan entre sí dentro del sistema.

4. Diagramas de Entidad Relación:



Otra manera:



LINK A MI GITHUB EN REPOSITORIO PUBLICO.

Entrega 2:

1. **Listado de VISTAS**
    1. **ventas_cliente** ○ Descripción: Vista que combina la información de clientes, ventas y productos, mostrando un desglose detallado de cada operación.

        ○ Objetivo: Consultar, para cada venta y cliente, los productos adquiridos, cantidad, precio unitario y subtotal.

        ○ Tablas: Cliente, Venta, Detalle_Venta, Producto.

    2. **stock_productos**

        ○ Descripción: Vista que lista todos los productos con su categoría, precio unitario y stock actual.

        ○ Objetivo: Acceder de manera rápida al inventario actualizado y organizado por categorías.

        ○ Tablas: Producto, Categoria

    3. **ventas_resumen_fecha**

        ○ Descripción: Vista que agrupa las ventas por fecha, mostrando la cantidad de ventas realizadas y el monto total vendido en cada día.

        ○ Objetivo: Obtener un resumen diario de ventas para reportes y control de gestión.

        ○ **Tablas: Venta, Detalle_Venta.**

    4. **compras_proveedor**

        ○ Descripción: Vista que muestra las compras realizadas a cada proveedor, incluyendo fecha, productos comprados, cantidades, precios y total de cada compra.

        ○ Objetivo: Consultar el historial de compras realizadas a proveedores con detalle de montos y fechas. ○ Tablas: Compra, Proveedor, Detalle_Compra.

    5. **ventas_por_categoria**

- **Descripción**: Vista que agrupa todas las ventas según la categoría de los productos vendidos, mostrando la cantidad total de productos vendidos por categoría y el monto total generado.

- **Objetivo**: Analizar qué categorías de productos generan más ingresos y volumen de ventas, facilitando decisiones sobre qué productos potenciar en el negocio.
- **Tablas**: Detalle_Venta, Producto, Categoria, Venta.
- **Beneficio**: Proporciona un resumen de ventas por categoría, útil para informes de gestión y para priorizar compras e inventario.

2. Listado de FUNCIONES

1. **calcular_total_venta(p_id_venta)**

- **Objetivo:** Esta función devuelve el total de una venta específica, calculado a partir del detalle de los productos vendidos. Se multiplica la cantidad de cada producto por su precio unitario y luego se suman los valores parciales.

- **Tablas involucradas:**
  - Detalle_Venta: se utilizan los campos cantidad y precio_unitario.

Detalle del proceso:
  - La función recibe como parámetro el id_venta.
  - Se filtran todos los registros en Detalle_Venta que correspondan a esa venta.
  - Se calcula el monto total sumando cantidad * precio_unitario de cada producto.

Caso de uso: SELECT calcular_total_venta(3) AS total_venta; Esto devuelve el monto total de la venta con ID = 3.

2. **calcular_total_compra(p_id_compra)**

- **Objetivo:** Esta función devuelve el total pagado en una compra a proveedor, calculando la suma de la cantidad de cada producto adquirido por su precio unitario.
- **Tablas involucradas:**
  - Detalle_Compra: se utilizan los campos cantidad y precio_unitario.

Detalle del proceso:
  - La función recibe como parámetro el id_compra.
  - Se filtran los registros en Detalle_Compra correspondientes a esa compra.
  - Se calcula el total mediante la suma de cantidad * precio_unitario.

Caso de uso:

```
SELECT calcular_total_compra(5) AS total_compra;
```

Esto devuelve el monto total de la compra con ID = 5.

3. Listado de STORE PROCEDURES

Primer sp: registrar_venta_json(p_id_cliente, p_fecha, p_items JSON)

- **Objetivo:** Registrar una nueva venta de forma completa a partir de un conjunto de productos en formato JSON. Este procedimiento inserta un registro en la tabla Venta, inserta todos los productos asociados en Detalle_Venta, descuenta automáticamente el stock de cada producto vendido y recalcula el total de la venta.
- **Beneficio:**
  - Permite registrar ventas complejas con múltiples productos en una sola operación.
  - Reduce la posibilidad de errores manuales al cargar detalles de venta uno por uno.
  - Integra automáticamente la lógica de actualización de stock y totales.

Tablas involucradas:

- Venta: se inserta la venta principal. ○ Detalle_Venta: se insertan los ítems del JSON.
- Producto: se actualiza el campo stock_actual.

Caso de uso:

```
CALL registrar_venta_json(
1,
'2025-09-09',
'[{"id_producto":1,"cantidad":2,"precio_unitario":100.50},
 {"id_producto":3,"cantidad":1,"precio_unitario":250.00}]'
);
```

Este sp hace que se genere una nueva venta para el cliente 1 en la fecha indicada, con dos productos (id 1 y 3). Automáticamente descuenta stock, inserta en detalle y actualiza el total en la tabla Venta.

Segundo sp: registrar_compra_json(p_id_proveedor, p_fecha, p_items JSON)

- **Objetivo:** Registrar una compra a proveedores en forma automática a partir de un conjunto de ítems en formato JSON. Inserta un registro en la tabla Compra, los detalles en Detalle_Compra, actualiza el stock de productos y recalcula el total de la compra.

- **Beneficio:**
    - Facilita la carga de compras masivas con varios productos en un solo paso.
    - Mantiene el stock sincronizado automáticamente con el inventario real.
    - Garantiza consistencia en el cálculo del total de la compra.
- **Tablas involucradas:**
    - Compra: se inserta el encabezado de la compra. ○ Detalle_Compra: se insertan los ítems del JSON.
    - Producto: se incrementa el stock_actual de cada producto comprado.
- **Caso de uso:**

```
CALL registrar_compra_json(
  2,
  '2025-09-09',
  '[{"id_producto":2,"cantidad":100,"precio_unitario":40.00},
   {"id_producto":5,"cantidad":50,"precio_unitario":120.00}]'
);
```

Similar al anterior, este sp genera una compra al proveedor 2 en la fecha indicada, con dos productos distintos. Automáticamente suma stock y actualiza el total en Compra.

4. Listado de TRIGGERS

1. **actualizar_stock_venta (AFTER INSERT ON Detalle_Venta)**
    i. **Objetivo:** Cada vez que se inserta un nuevo registro en Detalle_Venta, este trigger disminuye automáticamente el stock del producto vendido en la tabla Producto. Así se asegura que el inventario se mantenga actualizado sin necesidad de hacerlo manualmente.
    ii. **Tablas involucradas:**
        1. Detalle_Venta: usa id_producto y cantidad.
        2. Producto: actualiza el campo stock_actual.
    iii. **Detalle del proceso:** Se toma la cantidad del producto recién vendido y luego, se resta esa

cantidad al stock actual del producto correspondiente.

iv. Caso de uso:

Si se inserta en Detalle_Venta: (id_venta=3, id_producto=10, cantidad=2, precio_unitario=500), automáticamente se descuenta 2 unidades del producto con id_producto=10 en la tabla Producto.

2. **actualizar_stock_compra (AFTER INSERT ON Detalle_Compra)**

i. Objetivo: Cuando se registra una compra de productos a un proveedor en Detalle_Compra, este trigger aumenta el stock de dichos productos en la tabla Producto. Esto refleja la reposición de inventario.

ii. Tablas involucradas:

a. Detalle_Compra: usa id_producto y cantidad.
b. Producto: actualiza el campo stock_actual.

iii. **Detalle del proceso:**

a. Se toma la cantidad de productos comprados.
b. Se suma esa cantidad al stock actual del producto correspondiente. iv. **Caso de uso:**

Si se inserta en Detalle_Compra: (id_compra=7, id_producto=5, cantidad=100, precio_unitario=50), el stock del producto con id_producto=5 aumenta en 100 unidades.

3. **refresh_total_venta (AFTER INSERT ON Detalle_Venta)**

i. **Objetivo:** Cada vez que se inserta un detalle de venta, este trigger recalcula el total de la venta en la tabla Venta, asegurando que siempre refleje la suma correcta de todos los productos vendidos. ii. **Tablas involucradas:**

a. Detalle_Venta: usa id_venta, cantidad y precio_unitario.
b. Venta: actualiza el campo total.

iii. **Detalle del proceso:** El trigger detecta un nuevo detalle de venta. Llama a la función calcular_total_venta(p_id_venta). Esta actualiza el campo total en la tabla Venta con el nuevo cálculo. iv. **Caso de uso:**

Si Venta.id_venta = 3 tenía un total de $2000, y se agrega un detalle con (cantidad=3, precio_unitario=400), el trigger recalcula automáticamente el total de la venta, quedando en $3200.

4. **refresh_total_compra (AFTER INSERT ON Detalle_Compra)**

   i. **Objetivo:** Este trigger recalcula el monto total de la compra cada vez que se inserta un detalle en Detalle_Compra, manteniendo actualizado el campo total_compra en la tabla Compra.

   ii. **Tablas involucradas:**

   a. Detalle_Compra: usa id_compra, cantidad y precio_unitario.

   b. Compra: actualiza el campo total_compra. iii. **Detalle del proceso:**

   a. Se inserta un nuevo producto comprado en el detalle. Luego, se ejecuta la función calcular_total_compra(p_id_compra). El Trigger lo que hace es actualizar el campo total_compra en la tabla Compra. iv.

   Caso de uso:

   Si Compra.id_compra = 5 tenía un total de $5000, y se agrega un detalle (cantidad=10, precio_unitario=300), el trigger recalcula el total, quedando en $8000.

Listado de Tablas

1. Tabla Categoria

- id_categoria: identificador único de la categoría (INT, PK, AUTO_INCREMENT).
- nombre_categoria: nombre de la categoría (VARCHAR(50), NOT NULL).

2. Tabla Producto

- id_producto: identificador único del producto (INT, PK, AUTO_INCREMENT).
- nombre: nombre del producto (VARCHAR(100), NOT NULL).
- descripcion: descripción del producto (VARCHAR(255), NULL).
- id_categoria: categoría del producto (INT, FK → Categoria.id_categoria).
- precio_unitario: precio por unidad (DECIMAL(10,2), NOT NULL, CHECK > 0).
- stock_actual: cantidad disponible en inventario (INT, NOT NULL, DEFAULT 0, CHECK >= 0).

3. Tabla Cliente

- id_cliente: identificador único del cliente (INT, PK, AUTO_INCREMENT).
- nombre: nombre del cliente (VARCHAR(50), NOT NULL).
- apellido: apellido del cliente (VARCHAR(50), NOT NULL).
- telefono: teléfono de contacto (VARCHAR(20), NULL).
- email: correo electrónico (VARCHAR(100), NULL).
- direccion: dirección física (VARCHAR(255), NULL).

4. Tabla Proveedor

- id_proveedor: identificador único del proveedor (INT, PK, AUTO_INCREMENT).
- nombre_proveedor: nombre o razón social (VARCHAR(100), NOT NULL).
- telefono: teléfono de contacto (VARCHAR(20), NULL).
- email: correo electrónico (VARCHAR(100), NULL).
- direccion: dirección física (VARCHAR(255), NULL).

5. Tabla Venta

- id_venta: identificador único de la venta (INT, PK, AUTO_INCREMENT).
- fecha_venta: fecha de la venta (DATE, NOT NULL).
- id_cliente: cliente que realizó la compra (INT, FK → Cliente.id_cliente).
- total: monto total de la venta (DECIMAL(10,2), NOT NULL, CHECK >= 0).
- estado: estado de la venta (ENUM('pendiente','pagada','cancelada'), DEFAULT 'pendiente').

6. Tabla Detalle_Venta

- id_detalle: identificador único del detalle de venta (INT, PK, AUTO_INCREMENT).
- id_venta: referencia a la venta (INT, FK → Venta.id_venta).
- id_producto: referencia al producto vendido (INT, FK → Producto.id_producto).
- cantidad: cantidad vendida (INT, NOT NULL, CHECK > 0).

- precio_unitario: precio por unidad en esa venta (DECIMAL(10,2), NOT NULL, CHECK > 0).

7. Tabla Compra

- id_compra: identificador único de la compra (INT, PK, AUTO_INCREMENT).
- id_proveedor: proveedor de la compra (INT, FK → Proveedor.id_proveedor).
- fecha_compra: fecha de la compra (DATE, NOT NULL).
- total_compra: monto total de la compra (DECIMAL(10,2), NOT NULL, CHECK >= 0).
- estado: estado de la compra (ENUM('pendiente','pagada','cancelada'), DEFAULT 'pendiente').

8. Tabla Detalle_Compra

- id_detalle_compra: identificador único del detalle de compra (INT, PK, AUTO_INCREMENT).
- id_compra: referencia a la compra (INT, FK → Compra.id_compra).
- id_producto: referencia al producto comprado (INT, FK → Producto.id_producto).
- cantidad: cantidad comprada (INT, NOT NULL, CHECK > 0).
- precio_unitario: precio por unidad pagado (DECIMAL(10,2), NOT NULL, CHECK > 0).

LINK A MI REPOSITORIO