UNIVERSIDAD AUTÓNOMA METROPOLITANA

UNIDAD CUAJIMALPA

LICENCIATURA EN TECNOLOGÍAS Y SISTEMAS DE INFORMACIÓN.

Base de datos

Base de datos de una tienda online

Profesor:
Dr.Guillermo Monroy Rodríguez

Alumno:
Mateo Dorantes Andres-2233030470

Introducción

Durante la presentación de este documento se verá lo realizado para la creación y diseño de una base de datos sobre una tienda online y cómo se cumplen las necesidades para su buen funcionamiento además de como se optimizó y se plantearon problemáticas que puede tener y cómo se resolvieron.

Objetivo general

El objetivo de este proyecto fue la demostración de los conocimientos adquiridos en la UEA de base de datos y cómo estos fueron implementados en el proceso de la relación de una base de datos con los requisitos especificados.

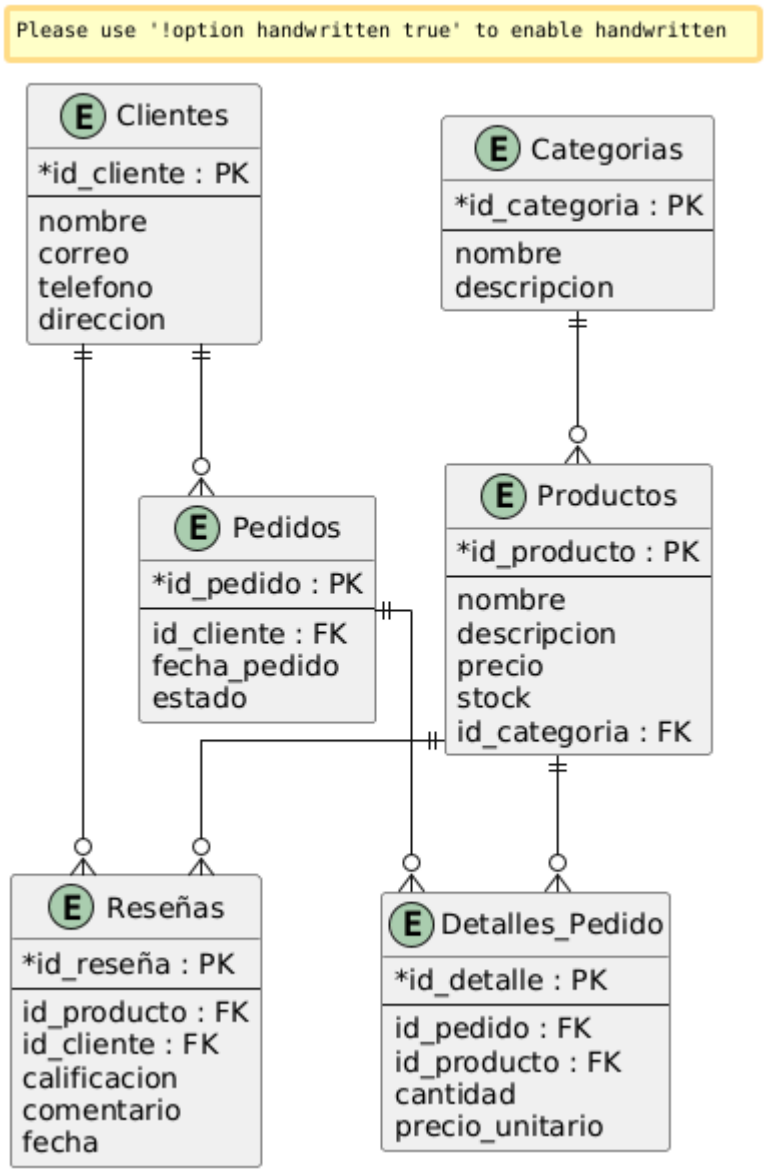
Análisis de la problemática

se requiere una base datos sobre una tienda online en donde se requieren los clientes , productos, categorías , reseñas, detalles de los pedidos , y pedido en base a estas tablas se realiza una normalización hasta la tercera forma normal además de implementación de claves primarias y secundarias para una mejor relación y evitar redundancias por lo tanto ya identificado lo que se tiene que hacer y los procedimientos como consultar almacenar o actualizar así como de mostrar datos en específico se realizan los paso para comenzar este proyecto

diseño del diagrama entidad relación

En base al análisis del problema y la identificación de las 6 tablas a trabar las cuales son(categoría,cliente, producto,reseña,pedidos y detalle pedido).
Se realizó el siguiente modelo entidad relación.

Figura 1(f1).



Esto realizado con el programa en línea plantUML en donde el código fuente se encontrará en el archivo Diagrama(ER).txt dentro de la carpeta de este proyecto.

Justificación de normalización

Modelo de Base de Datos: Normalización hasta la Tercera Forma Normal (3NF)

Este modelo de base de datos se ha diseñado de acuerdo con la **3NF**, lo que asegura que:

- Cada tabla representa una sola entidad.
- Todos los atributos no clave dependen directamente de la clave primaria.
- Se minimizan las dependencias transitivas y la redundancia de datos.
- Se asegura consistencia, eficiencia y facilidad de mantenimiento.

Tablas y Atributos

Tabla: Clientes

- **id_cliente** (PK): Identificador único del cliente.
- **Nombre** del cliente.
- **correo**: Dirección de correo electrónico del cliente (debe ser única).
- **telefono**: Número de teléfono del cliente.
- **direccion**: Código postal del cliente.

Tabla: Productos

- **id_producto** (PK): Identificador único del producto.
- **nombre**: Nombre del producto.
- **descripcion**: Descripción detallada del producto.
- **precio**: Precio unitario del producto.
- **stock**: Cantidad disponible en inventario (no debe ser negativa).
- **id_categoria** (FK → Categorías.id_categoria): Categoría a la que pertenece el producto.

Tabla: Categorías

- **id_categoria** (PK): Identificador único de la categoría.
- **nombre**: Nombre de la categoría de los productos.
- **descripcion**: Descripción general de la categoría.

Tabla: Pedidos

- **id_pedido** (PK): Identificador único del pedido.
- **id_cliente** (FK → Clientes.id_cliente): Cliente que realizó el pedido.
- **fecha_pedido**: Fecha en que se realizó el pedido.
- **estado**: Estado actual del pedido (pendiente, enviado, entregado).

Tabla: Detalles_Pedido

- **id_detalle** (PK): Identificador único del detalle del pedido.
- **id_pedido** (FK → Pedidos.id_pedido): Pedido perteneciente al detalle.
- **id_producto** (FK → Productos.id_producto): Producto al que pertenece este detalle.
- **cantidad**: Cantidad del producto solicitada en el pedido.
- **precio_unitario**: Precio del producto en el momento del pedido.

Tabla: Reseñas

- **id_resena** (PK): Identificador único de la reseña.
- **id_producto** (FK → Productos.id_producto): Producto al que se refiere la reseña.
- **id_cliente** (FK → Clientes.id_cliente): Cliente que escribió la reseña.
- **calificacion**: Calificación del producto (de 1 a 5 estrellas).
- **comentario**: Comentario escrito por el cliente sobre el producto.
- **fecha**: Fecha en que se publicó la reseña.

Justificación de la Normalización a 3NF

General

El diseño propuesto cumple con la **Tercera Forma Normal (3NF)**. Esto significa que cada atributo no clave depende directamente de la clave primaria de su tabla. Además, se evitan dependencias transitivas y redundancia de datos, lo que garantiza integridad y eficiencia.

Justificación por Tabla

Clientes (id_cliente, nombre, correo, telefono, direccion)

- Todos los atributos dependen directamente de id_cliente.

Categorías (id_categoria, nombre, descripcion)

- Los atributos nombre y descripcion dependen directamente de id_categoria.
- Cada categoría representa una entidad independiente.

Productos (id_producto, nombre, descripcion, precio, stock, id_categoria)

- Todos los atributos dependen de id_producto.
- id_categoria es una clave foránea que evita la duplicación de datos relacionados con la categoría en esta tabla.

Pedidos (id_pedido, id_cliente, fecha_pedido, estado)

- Los atributos fecha_pedido y estado dependen directamente de id_pedido.
- id_cliente es una clave foránea que enlaza con Clientes, sin repetir información del cliente.

Detalles_Pedido (id_detalle, id_pedido, id_producto, cantidad, precio_unitario)

- La tabla actúa como una relación de muchos a muchos entre Pedidos y Productos.
- cantidad y precio_unitario dependen directamente de id_detalle (o de la clave compuesta id_pedido + id_producto).

Reseñas (id_reseña, id_producto, id_cliente, calificacion, comentario, fecha)

- Los campos calificacion, comentario y fecha dependen directamente de id_resena.
- id_producto y id_cliente son claves foráneas que conectan con Productos y Clientes, evitando duplicación de datos.

Conclusión

La normalización de la **3NF** nos permite la minimización de la redundancia, a su vez evitar complicaciones en inserciones, actualizaciones y eliminaciones, poder facilitar la integridad referencial entre tablas y por último optimizar la estructura para mantenimiento y escalabilidad.

Creación de script para tablas

Crear Base de Datos

```
CREATE DATABASE tienda_online;
USE tienda_online;
```

Aquí se crea la base dedatos con su nombre y posteriormente se utiliza para crear las tablas dentro de esta misma

Tabla Categorías

```
CREATE TABLE Categorias (
    id_categoria INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL UNIQUE,
    descripcion TEXT
);
```

En esta tabla se crea su estructura en donde tiene una primero key auto incrementable un nombre con una longitud de 100 caracteres la cual no debe ser vacía y tiene que ser unacia para evitar el duplicado de categorias y una descripcion con tipo de dato texto ya que la longitud de caracteres a +poner es larga y variable entre otras

Tabla Clientes

```
CREATE TABLE Clientes (
    id_cliente INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    correo VARCHAR(255) NOT NULL UNIQUE,
    telefono VARCHAR(20),
```

```
    direccion VARCHAR(255)
);
```

En esta tabla se crean los valores básicos que debe tener un cliente con su respectiva clave primaria autoincrementable donde el nombre y correo no pueden ser nulos ya que son datos indispensables y el correo debe ser único para evitar tener correos diferentes del mismo cliente.

Tabla Productos

```
CREATE TABLE Productos (
    id_producto INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    descripcion TEXT,
    precio DECIMAL(12, 2) NOT NULL CHECK (precio >= 0),
    stock INT NOT NULL CHECK (stock >= 0),
    id_categoria INT NOT NULL,
    FOREIGN KEY (id_categoria) REFERENCES Categorias(id_categoria),
    UNIQUE (nombre, id_categoria)
);
```

Dentro de esta tabla se establece el primary key, los datos básicos de un producto como se relaciona con categoría los datos utilizados tiene los mismo propósitos que los anteriores por su parte precio es decimal ya que ocupa tener el apartado de centavos de los dos primeros siendo 12 el total de casillas y 2 las callias que va a tomar para los flotantes, dejan 10 casillas para los enteros por si hay productos caros en este caso al igual que el stock se pone una restricción de tipo check en donde esta tiene la función de darle una condición al dato que estamos utilizando en este caso que tanto precio como producto no son negativos y tenga un valor igual o mayor a cero resolviendo así una de las problemáticas y se hace una referencia a id categoría para darle un tipo de categoría al producto pero asu ves que no se repita el mismo nombre del producto en esa categoría.

Tabla Pedidos

```
CREATE TABLE Pedidos (
    id_pedido INT AUTO_INCREMENT PRIMARY KEY,
    id_cliente INT NOT NULL,
    fecha_pedido DATE NOT NULL,
        estado ENUM('pendiente', 'enviado', 'entregado', 'cancelado') NOT NULL
DEFAULT 'pendiente',
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
);
```

En esta tabla se hace una relación con el cliente y una clave primaria para el pedido,dentro del cau se guarda la fecha ingresada cuando se realizó en pedido,además se tiene el estado en el que se encuentra el pedido utilizando en este caso en tipo de restricción en el cual es un tipo de dato en el queso puede personalizar cual es la lista de datos en la que se encasilla la variable y sólo pueda utilizarse lo que está dentro de esta en este caso sería pendiente, enviado, entregado y cancelado) y dejando por defecto el estado del pedido en pendiente.

Tabla Detalles_Pedido

```
CREATE TABLE Detalles_Pedido (
    id_detalle INT AUTO_INCREMENT PRIMARY KEY,
    id_pedido INT NOT NULL,
    id_producto INT NOT NULL,
    cantidad INT NOT NULL CHECK (cantidad > 0),
    precio_unitario DECIMAL(12, 2) NOT NULL,
    FOREIGN KEY (id_pedido) REFERENCES Pedidos(id_pedido),
    FOREIGN KEY (id_producto) REFERENCES Productos(id_producto),
    UNIQUE (id_pedido, id_producto)
);
```

En base a la estructura que se tiene se puede describir que se realiza un a relación con pedidos y productos se obtiene su propia primary key que la cantidad tiene una restricción de tipo chec donde debe ser mayor a cero ya que pues no existiera un detalle de un pedido si se tiene cero pedidos y el precio se maneja de la misma forma que el precio de productos donde por último se maneja un unique para que no exista una redundancia de de detalles medios para evitar que se agran pedidos repetidos del mismo producto por parte del cliente

Tabla Resenas

```
CREATE TABLE Resenas (
    id_resena INT AUTO_INCREMENT PRIMARY KEY,
    id_producto INT NOT NULL,
    id_cliente INT NOT NULL,
    calificacion INT NOT NULL CHECK (calificacion >= 1 AND calificacion <= 5),
    comentario TEXT,
    fecha DATE NOT NULL,
    FOREIGN KEY (id_producto) REFERENCES Productos(id_producto),
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
);
```

En esta tabla de reseñas se estructura de esta forma teniendo una relación con productos y cliente para saber qué cliente hace la reseña y aq ue producto además de su clave primaria de esta tabla como a su vez en la variable calificación se le pone una restricción de tipo check para que solo se pueden poner calificación del rango de uno a cinco puntos y el comentario es de tipo text ya que es muy incierto definir una limitante de caracteres por varchar ya que es muy incierto la cantidad de lo que se escribe en un comentario y subes la fecha se maneja de la misma forma que la anterior mencionada en tabla pedidos.

4. Índices Creados

```
CREATE INDEX idx_productos_nombre_categoria ON Productos (nombre, id_categoria);
CREATE INDEX idx_pedidos_cliente ON Pedidos (id_cliente);
CREATE INDEX idx_resenas_producto ON Resenas (id_producto);
CREATE INDEX idx_resenas_cliente ON Resenas (id_cliente);
```

se crearon estos índices para acelerar consultas o búsquedas comunio como:

- Búsqueda de productos por nombre y categoría (idx_productos_nombre_categoria).
- Pedidos de un cliente (idx_pedidos_cliente).
- Reseñas por producto (idx_resenas_producto) o por cliente (idx_resenas_cliente).

Creación de poblador

Propósito del script de Inserción de Datos de Prueba

Este script está diseñado para **insertar datos de prueba (mock data)** en las tablas previamente creadas de la base de datos Tienda_Online.

Estructura y Contenido del Script

Seleccionar la base de datos

USE Tienda_Online;

Poblamiento de la Tabla Categorias

INSERT INTO Categorias (nombre, descripcion) VALUES ...

Se insertan cuatro categorías generales: Electrónica, Hogar, Moda y Libros. Estas categorías se utilizarán para clasificar los productos en la base de datos.

Poblamiento de la Tabla Clientes

INSERT INTO Clientes (...) VALUES ...

Se insertan 15 clientes con sus respectivos datos(nombre, correo, teléfono y dirección). Esta información es útil para las compras, pedidos y reseñas de los usuarios.

Poblamiento de la Tabla Productos

INSERT INTO Productos (...) VALUES ...

Se insertan 30 productos distribuidos en las cuatro categorías existentes. Cada producto incluye descripciones, precios, cantidades en stock y su relación con una categoría específica.

Poblamiento de la Tabla Pedidos

INSERT INTO Pedidos (...) VALUES ...

Se insertan 20 pedidos realizados por diferentes clientes, cada uno con su fecha y estado (pendiente, enviado, entregado y cancelado).

Poblamiento de la Tabla Detalles_Pedido

INSERT INTO Detalles_Pedido (...) VALUES ...

Se insertan 25 detalles de pedidos que especifican qué productos se compraron en cada pedido, la cantidad y el precio unitario en el momento de la compra.

7. Poblamiento de la Tabla Reseñas

INSERT INTO Reseñas (...) VALUES ...

Se insertan **10 reseñas** de clientes sobre productos. Cada reseña incluye una calificación (de 1 a 5 estrellas), un comentario y la fecha de publicación.Mostrar valoraciones promedio de productos.

Nota: el script completo de este apartado se puede consulta en en este proyecto el cual se llama poblador.sql no se colocó de manera completa por la gran cantidades de datos que se encuentran en este mismo y por locanto haría innecesariamente largo el documento por lo cual se optó por la descripción clara de este mismo.

Conclusión

Este script proporciona un conjunto robusto de datos, ideal para validar el funcionamiento del sistema Tienda_Online.

Creación de consultas.

Consulta 1 Productos disponibles por categoría

```
SELECT
    p.nombre AS nombre_producto,
```

```
    p.precio,
    p.stock,
    c.nombre AS nombre_categoria
FROM Productos AS p
INNER JOIN Categorías AS c ON p.id_categoria = c.id_categoria
WHERE p.stock > 0
ORDER BY c.nombre, p.precio;
```

Con esta consulta se puede mostrar todos los productos actualmente disponibles (con stock > 0), junto con su categoría donde esta información está ordenada primero por nombre de categoría y luego por precio del producto.

Justificación de la estructura:

Se usa un INNER JOIN entre Productos y Categorías para obtener el nombre de la categoría, en donde la condición WHERE p.stock > 0 se usa para que muestre los productos con stock disponible mayor a cero y ORDER BY hace un ordenamiento por catalogo.

Consulta 2 Clientes con pedidos pendientes y total de compras

```
SELECT
    c.id_cliente,
    c.nombre AS nombre_cliente,
    SUM(dp.cantidad * dp.precio_unitario) AS total_compras
FROM Clientes AS c
INNER JOIN Pedidos AS p ON c.id_cliente = p.id_cliente
INNER JOIN Detalles_Pedido AS dp ON p.id_pedido = dp.id_pedido
WHERE p.estado = 'pendiente'
GROUP BY c.id_cliente, c.nombre
ORDER BY total_compras DESC;
```

En base a la consulta podemos obtener una lista de clientes que tienen pedidos pendientes, junto con el total acumulado de sus compras en esos pedidos.

Justificación de la estructura:

Se utiliza INNER JOIN para relacionar Clientes, Pedidos y Detalles_Pedido, además se realiza un filtrado de solo los pedidos con el estado 'pendiente' esto especificado en el where., se agrupó por cliente para obtener sus compras en una sola línea.

SUM() permite calcular el valor de los precios de los productos en cada pedido.posteriormente se ordena de manera descendente por total_compras para mostrar a los clientes que más deben esto con el tipo de dato DESC.

Consulta 3 Top 5 productos mejor calificados

```
SELECT
    p.id_producto,
    p.nombre AS nombre_producto,
    AVG(r.calificacion) AS calificacion_promedio
FROM Productos AS p
INNER JOIN Resenas AS r ON p.id_producto = r.id_producto
GROUP BY p.id_producto, p.nombre
ORDER BY calificacion_promedio DESC
LIMIT 5;
```

Con esta consulta se genera un ranking de los 5 productos mejor calificados en base al promedio de las reseñas de los clientes.

Justificación de la estructura:

> Se utiliza INNER JOIN entre Productos y Resenas para obtener las evaluaciones., luego con AVG() se utiliza para calcular el promedio de las calificaciones. Posteriormente se agrupa por por producto para solo tener un producto en la línea en el caso de que se repita dentro de las calificaciones y con DESC se ordena de mayor a menor en el caso de que se quiera hacer de menor a mayor se utiliza ASCy por último se usa el tipo de datos LIMIT 5 para solo mostrar los 5 mejores del grupo.

Creación de procedimientos

En este apartado se observará la estructura de la creación de 8 procedimientos almacenados para crear procesos y su uso dentro de la base de datos de Tienda_Online agilizando las consultas o procesos más comunes.

RegistrarPedido

```
DELIMITER $$
CREATE PROCEDURE RegistrarPedido(
    IN p_id_cliente INT,
    IN p_id_producto INT,
    IN p_cantidad INT
)
BEGIN
    DECLARE v_stock_actual INT;
    DECLARE v_pedidos_pendientes INT;

    SELECT COUNT(*) INTO v_pedidos_pendientes
    FROM Pedidos
    WHERE id_cliente = p_id_cliente AND estado = 'pendiente';

    IF v_pedidos_pendientes >= 5 THEN
        SIGNAL SQLSTATE '45000'
          SET MESSAGE_TEXT = 'El cliente tiene 5 o más pedidos pendientes. No se
puede registrar un nuevo pedido.';
    ELSE
        SELECT stock INTO v_stock_actual
        FROM Productos
        WHERE id_producto = p_id_producto;

        IF v_stock_actual >= p_cantidad THEN
            INSERT INTO Pedidos (id_cliente, estado)
            VALUES (p_id_cliente, 'pendiente');

            SET @last_pedido_id = LAST_INSERT_ID();

                    INSERT INTO Detalles_Pedido (id_pedido, id_producto, cantidad,
precio_unitario)
            SELECT @last_pedido_id, id_producto, p_cantidad, precio
            FROM Productos
            WHERE id_producto = p_id_producto;

            CALL ActualizarStockProducto(p_id_producto, p_cantidad);

            SELECT 'Pedido registrado exitosamente.' AS Mensaje;
```

```
        ELSE
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Stock insuficiente para el producto solicitado.';
        END IF;
    END IF;
END$$
DELIMITER ;
```

Este procedimiento registra un nuevo pedido para un cliente, validando dos condiciones, que el cliente no puede tener más de cinco pedidos pendientes y tener el suficiente stock.

Funcionalidad:

1. Valida el número de pedidos pendientes del cliente.
2. Verifica la disponibilidad de stock del producto.
3. Si las validaciones son exitosas, registra el nuevo pedido en la tabla Pedidos.
4. Inserta el detalle del producto y la cantidad en la tabla Detalles_Pedido.
5. Llama al procedimiento ActualizarStockProducto para reducir el inventario.

RegistrarReseña

```
DELIMITER $$
CREATE PROCEDURE RegistrarReseña(
    IN p_id_producto INT,
    IN p_id_cliente INT,
    IN p_calificacion TINYINT,
    IN p_comentario TEXT
)
BEGIN
    DECLARE v_ha_comprado INT;

    SELECT COUNT(*) INTO v_ha_comprado
    FROM Detalles_Pedido AS dp
    JOIN Pedidos AS p ON dp.id_pedido = p.id_pedido
    WHERE dp.id_producto = p_id_producto AND p.id_cliente = p_id_cliente;

    IF v_ha_comprado > 0 THEN
```

```
        INSERT INTO Reseñas (id_producto, id_cliente, calificacion, comentario)
        VALUES (p_id_producto, p_id_cliente, p_calificacion, p_comentario);

        SELECT 'Reseña registrada exitosamente.' AS Mensaje;
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'El cliente no ha comprado este producto.';
    END IF;
END$$
DELIMITER ;
```

El procedimiento permite registrar una reseña sobre un producto, pero solo si el cliente ya ha comprado dicho producto.

Funcionalidad:

1. Valida si el cliente ha comprado el producto, consultando si existe al menos un Detalle_Pedido con ese producto y cliente.
2. Si lo ha comprado entonces inserta una nueva reseña con calificación, comentario y fecha actual además de devolver un mensaje de éxito.
3. Si no ha comprado el producto, lanza un error para evitar reseñas inválidas.

ActualizarStockProducto

```
DELIMITER $$
CREATE PROCEDURE ActualizarStockProducto(
    IN p_id_producto INT,
    IN p_cantidad INT
)
BEGIN
    UPDATE Productos
    SET stock = stock - p_cantidad
    WHERE id_producto = p_id_producto AND stock >= p_cantidad;
END$$
DELIMITER ;
```

Con este procedimiento se reduce la cantidad de stock de un producto en la base de datos. Este procedimiento es utilizado en el primer procedimiento al registrar un pedido para descontar los pedidos registrados en este.

Funcionalidad:

1. Actualiza el stock del producto con `id_producto`, restándole la cantidad indicada (`stock = stock - p_cantidad`).
2. Solo se ejecuta si el stock actual es suficiente (evita stock negativo).
3. No lanza error si no se cumple la condición ya que no realiza el procedimiento y no afecta la base de datos.

CambiarEstadoPedido

```
DELIMITER $$
CREATE PROCEDURE CambiarEstadoPedido(
    IN p_id_pedido INT,
    IN p_nuevo_estado ENUM('pendiente', 'enviado', 'entregado', 'cancelado')
)
BEGIN
    UPDATE Pedidos
    SET estado = p_nuevo_estado
    WHERE id_pedido = p_id_pedido;
END$$
DELIMITER ;
```

El procedimiento permite cambiar el estado de un pedido existente a uno de los estados predefinidos con anterioridad: 'pendiente', 'enviado', 'entregado' o 'cancelado'.

Funcionalidad:

1. Recibe el `id_pedido` y un nuevo estado válido (`'pendiente'`, 'enviado', 'entregado', 'cancelado').
2. Actualiza directamente el campo estado del pedido.

EliminarReseñasProducto

```
DELIMITER $$
CREATE PROCEDURE EliminarReseñasProducto(
    IN p_id_producto INT
)
BEGIN
    DELETE FROM Reseñas
    WHERE id_producto = p_id_producto;

    SELECT 'Reseñas eliminadas exitosamente.' AS Mensaje;
END$$
DELIMITER ;
```

Elimina todas las reseñas que estén asociadas a un producto.

Funcionalidad:

1. Borra todas las reseñas asociadas al producto con base al id_producto.
2. Se ejecuta un DELETE FROM Reseñas para ese producto.
3. Devuelve un mensaje indicando que las reseñas fueron eliminadas correctamente.

AgregarNuevoProducto

```
DELIMITER $$
CREATE PROCEDURE AgregarNuevoProducto(
    IN p_nombre VARCHAR(255),
    IN p_descripcion TEXT,
    IN p_precio DECIMAL(10, 2),
    IN p_stock INT,
    IN p_id_categoria INT
)
BEGIN
    DECLARE v_duplicado INT;

    SELECT COUNT(*) INTO v_duplicado
    FROM Productos
    WHERE nombre = p_nombre AND id_categoria = p_id_categoria;
```

```
    IF v_duplicado = 0 THEN
        INSERT INTO Productos (nombre, descripcion, precio, stock, id_categoria)
        VALUES (p_nombre, p_descripcion, p_precio, p_stock, p_id_categoria);

        SELECT 'Producto agregado exitosamente.' AS Mensaje;
    ELSE
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Ya existe un producto con el mismo nombre y categoría.';
    END IF;
END$$
DELIMITER ;
```

El procedimiento agrega un nuevo producto al inventario, con una validación para asegurar que no se duplique un producto con el mismo nombre dentro de la misma categoría.

Funcionalidad:

1. Verifica si ya existe un producto con el mismo nombre en la misma categoría (UNIQUE(nombre, id_categoria)).
2. Si no existe duplicado, inserta el nuevo producto con sus atributos (nombre, descripcion, precio, stock, id_categoria).
3. Si ya existe uno igual, lanza un error.

ActualizarCliente

```
DELIMITER $$
CREATE PROCEDURE ActualizarCliente(
    IN p_id_cliente INT,
    IN p_nombre VARCHAR(100),
    IN p_telefono VARCHAR(20),
    IN p_direccion VARCHAR(255)
```

```
)
BEGIN
    UPDATE Clientes
    SET nombre = p_nombre,
        telefono = p_telefono,
        direccion = p_direccion
    WHERE id_cliente = p_id_cliente;
END$$
DELIMITER ;
```

Este procedimiento actualiza la información básica de un cliente existente, como su nombre, teléfono y dirección.

Funcionalidad:

1. Recibe el id_cliente junto con los nuevos valores de nombre, telefono y direccion.
2. Actualiza la información del cliente en la tabla Clientes.

ReporteStockBajo

```
DELIMITER $$
CREATE PROCEDURE ReporteStockBajo()
BEGIN
    SELECT
        nombre AS nombre_producto,
        stock
    FROM Productos
    WHERE stock < 5
    ORDER BY stock ASC;
END$$
DELIMITER ;
```

Con este procedimiento se genera un listado de todos los productos cuyo stock actual es inferior a cinco unidades, ordenándolos por la cantidad de stock de forma ascendente.

Funcionalidad:

1. Realiza una consulta en la tabla Productos para obtener todos los productos cuyo stock sea menor a 5.
2. Ordena los resultados de menor a mayor para facilitar la identificación de productos más bajos.

Fase e informe de pruebas

Sentencias sql

Consulta 1 Productos disponibles por categoría

```
mysql> source Sentencias.sql;
Database changed
Query OK, 0 rows affected (0.003 sec)

+-------------------------------+----------+-------+------------------+
| nombre_producto               | precio   | stock | nombre_categoria |
+-------------------------------+----------+-------+------------------+
| Auriculares Inalambricos      |   959.00 |   150 | Electronica      |
| Smartphone S20                | 12002.99 |    50 | Electronica      |
| Laptop Gamer X1               | 14999.50 |    25 | Electronica      |
| Smartwatch FitPro             | 15901.00 |    75 | Electronica      |
| Televisor 4K 57"              | 65000.00 |    30 | Electronica      |
| Vaso termico 500ml            |   155.00 |   300 | Hogar            |
| Juego de toallas              |   365.00 |   200 | Hogar            |
| Cafetera programable          |   454.00 |    80 | Hogar            |
| Sarten Antiadherente          |   552.00 |   120 | Hogar            |
| Licuadora 12 velocidades      |   705.50 |   100 | Hogar            |
| Juego de cuchillos profesional|   850.99 |    60 | Hogar            |
| Microondas con grill          |  1120.00 |    70 | Hogar            |
| Robot aspirador               |  2570.00 |    50 | Hogar            |
| Robot de cocina               |  2998.99 |    40 | Hogar            |
| Guia de viajes por Europa     |    22.00 |   100 | Libros           |
| Programaci├│n en Python        |    30.00 |    75 | Libros           |
| Novela de misterio            |   138.50 |   200 | Libros           |
| Libro de cocina italiana      |   225.00 |   150 | Libros           |
| Historia de Mexico            |   268.00 |    90 | Libros           |
| El Senor de los Anillos       |   652.00 |    50 | Libros           |
| Calcetines de algodon         |    15.00 |   500 | Moda             |
| Corbata de seda               |    25.00 |   100 | Moda             |
| Gafas de sol                  |    30.00 |   250 | Moda             |
| Cinturon de cuero rojo        |    30.00 |   120 | Moda             |
| Bufanda de lino               |    40.00 |    80 | Moda             |
| Camisa de Lino Azul           |    45.00 |    90 | Moda             |
| Falda corta                   |    50.00 |    60 | Moda             |
| Vestido de verano floreado    |    55.00 |    70 | Moda             |
| Pantalon Jeans overside       |    60.00 |   110 | Moda             |
| Zapatillas Deportivas         |    85.00 |   150 | Moda             |
| Bolso de piel sintetico       |   120.00 |    50 | Moda             |
+-------------------------------+----------+-------+------------------+
31 rows in set (0.268 sec)
```

Consulta 2 Clientes con pedidos pendientes y total de compras

```
Query OK, 0 rows affected (0.004 sec)

+------------+-----------------+---------------+
| id_cliente | nombre_cliente  | total_compras |
+------------+-----------------+---------------+
|         12 | Daniela Vargas  |        899.99 |
|          4 | Carlos Ruiz     |        195.00 |
|          9 | Fernanda Castro |         95.00 |
+------------+-----------------+---------------+
3 rows in set (0.074 sec)
```

Consulta 3 Top 5 productos mejor calificados

```
Query OK, 0 rows affected (0.003 sec)

+-------------+--------------------------+-----------------------+
| id_producto | nombre_producto          | calificacion_promedio |
+-------------+--------------------------+-----------------------+
|           1 | Smartphone S20           |                5.0000 |
|           2 | Laptop Gamer X1          |                5.0000 |
|          27 | Libro de cocina italiana |                5.0000 |
|           6 | Licuadora 12 velocidades |                4.0000 |
|          10 | Robot de cocina          |                4.0000 |
+-------------+--------------------------+-----------------------+
5 rows in set (0.025 sec)

mysql>
```

se puede ver que cada sentencia muestra los datos esperados lo cual muestra su buen funcionamiento

Procedimientos sql

procedimientos ejecutados correctamente

```
Query OK, 0 rows affected (0.004 sec)

Query OK, 1 row affected (0.104 sec)

Query OK, 0 rows affected (0.005 sec)

Query OK, 0 rows affected (0.014 sec)

Query OK, 0 rows affected (0.003 sec)

+---------------------------------+
| Mensaje                         |
+---------------------------------+
| Resenas eliminadas exitosamente. |
+---------------------------------+
1 row in set (0.014 sec)

Query OK, 0 rows affected (0.067 sec)

Query OK, 0 rows affected (0.004 sec)

+---------------------------------+
| Mensaje                         |
+---------------------------------+
| Producto agregado exitosamente. |
+---------------------------------+
1 row in set (0.084 sec)

Query OK, 0 rows affected (0.123 sec)

Query OK, 0 rows affected (0.005 sec)

Query OK, 1 row affected (0.081 sec)

Query OK, 0 rows affected (0.005 sec)

Empty set (0.017 sec)

Query OK, 0 rows affected (0.023 sec)
```

Aquí se puede observar la correcta ejecución de los procedimientos en base la base de datos donde los call en el archivo llamado procedimiento.sql que se puede consultar en este archivo además se muestran los mensajes de señal que se tienen.

procedimientos ejecutados con errores

```
ERROR 1644 (45000): Stock insuficiente para el producto solicitado.
```

Este primer error es mostrado cuando no se tiene un stock suficiente para realizar un pedido

```
ERROR 1644 (45000): El cliente tiene 5 o más pedidos pendientes. No se puede registrar un nuevo pedido.
```

Aquí se muestra el error para cuando el cliente supera los 5 pedidos máximos como pendientes para evitar muchos pedidos acomulados y sobre saturar el sistema

```
ERROR 1644 (45000): El cliente no ha comprado este producto.
```

En este error se muestra cuando se quiere insertar una nueva reseña pero el cliente ni a comparado el producto evitando malas o buenas valoraciones falsas

```
ERROR 1064 (42000): You have an error in your SQL syntax; check
 for the right syntax to use near '[redacted]asProducto(12)' at line 1
```

Este es solo un un error de sintaxis al tratado de borrar un producto escribiendo y usando el procedimiento de un manera errónea esto es más un error que se muestra en el mysql

Por último al tratar de insertar un nuevo producto manda este error al ver que ya existe y evitar redundancia de datos repetidor verificando si ya existe tanto el producto en la tabla productos y dentro de esa categoría

índices y su impacto

Los índices son estructuras que se crean en una o más columnas de una tabla para mejorar la velocidad de las operaciones de búsqueda y recuperación de datos. Funcionan de manera similar a un índice de libro, permitiendo al sistema de base de datos encontrar información rápidamente sin tener que revisar cada registro.

1. idx_productos_nombre_categoria en Productos (nombre, id_categoria)

   Este índice agiliza la búsqueda por nombre y o la categoría del producto además, mejora la velocidad de búsquedas como SELECT * FROM Productos WHERE nombre = 'Smartphone S20'.

2. idx_pedidos_cliente en Pedidos (id_cliente)

   Índice para consultas que filtran id cliente con la tabla pedidos agilizando la consulta de pedidos por cliente como SELECT * FROM Pedidos WHERE id_cliente = 5 y las uniones con la tabla Clientes.

3. idx_resenas_producto en Resenas (id_producto)

   El índice optimiza la búsqueda de las reseñas por productos las cuales podrian ser como SELECT calificacion, comentario FROM Resenas WHERE id_producto = 1 y las uniones con la tabla Productos.

4. idx_resenas_cliente en Resenas (id_cliente)

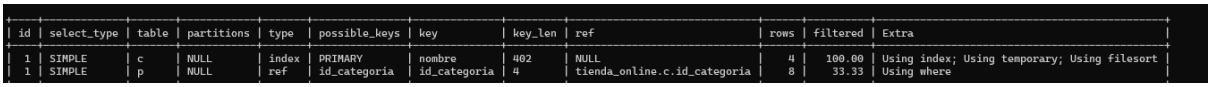
   Índice para las busquedas de la relación reseña cliente celebrando la visualización de que cliente realiza esta reseña la cual puede ser como SELECT id_producto, calificacion FROM Resenas WHERE id_cliente = 1 y las uniones con la tabla Clientes.

Impacto General de los Índices

El impacto que tienen dentro de esta base de datos es el mejoramiento de consultas así como su reducción de tiempo de ejecución además de de poder filtrar o ordenar los campos que estas manejas. Por otro lado los índices si ocupan un espacio dentro del almacenamiento del dispositivo lo que impacta a la ejecución de las otras sentencias normales(insert,update,select y delete)

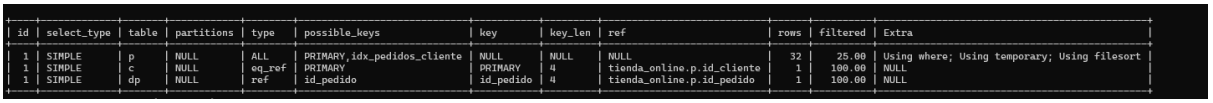
Uso de expain

Estos son los cuatro resultados hechos con el explain.sql que se encuentra dentro de este proyecto.



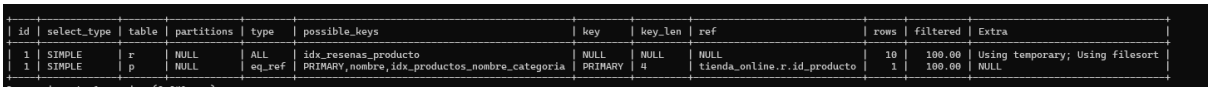
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	index	PRIMARY	nombre	402	NULL	4	100.00	Using index; Using temporary; Using filesort
1	SIMPLE	p	NULL	ref	id_categoria	id_categoria	4	tienda_online.c.id_categoria	8	33.33	Using where

Explain 1 (Productos con Stock y Ordenación): Eficiente en uniones, ineficiente en ordenación. Lo más importante aquí es que, si bien la consulta usa índices para unir tablas (Categorias y Productos), la necesidad de ordenar por dos columnas (c.nombre, p.precio) obliga a MySQL a crear una tabla temporal y ordenar los resultados en memoria/disco (Using temporary; Using filesort). Esto puede ser un cuello de botella con muchas filas.



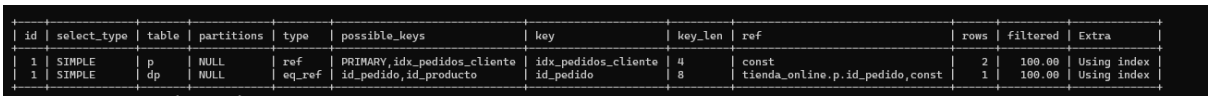
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	NULL	ALL	PRIMARY,idx_pedidos_cliente	NULL	NULL	NULL	32	25.00	Using where; Using temporary; Using filesort
1	SIMPLE	c	NULL	eq_ref	PRIMARY	PRIMARY	4	tienda_online.p.id_cliente	1	100.00	NULL
1	SIMPLE	dp	NULL	ref	id_pedido	id_pedido	4	tienda_online.p.id_pedido	1	100.00	NULL

Explain 2 (Total de Compras por Cliente con Pedidos Pendientes): Gran cuello de botella en el filtrado inicial. El punto crítico es el escaneo completo de la tabla Pedidos (type: ALL) para filtrar por estado = 'pendiente'** aunque existe un índice posible. Esto significa que MySQL lee cada pedido, lo cual es muy ineficiente. La solución más importante sería **crear un índice compuesto en Pedidos(estado, id_cliente)` para acelerar drásticamente el filtrado y la agrupación.



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	r	NULL	ALL	idx_resenas_producto	NULL	NULL	NULL	10	100.00	Using temporary; Using filesort
1	SIMPLE	p	NULL	eq_ref	PRIMARY,nombre,idx_productos_nombre_categoria	PRIMARY	4	tienda_online.r.id_producto	1	100.00	NULL

Explain 3 (Top 5 Productos por Calificación Promedio): Escaneo completo de reseñas, afecta el rendimiento. Similar a la Consulta 2, el problema principal es el escaneo completo de la tabla Resenas (type: ALL) para calcular las calificaciones promedio. A pesar de tener un índice en id_producto, no se utiliza para el escaneo inicial. Un índice en Resenas(id_producto, calificacion) es crucial para mejorar la eficiencia del cálculo y la agrupación.



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	p	NULL	ref	PRIMARY,idx_pedidos_cliente	idx_pedidos_cliente	4	const	2	100.00	Using index
1	SIMPLE	dp	NULL	eq_ref	id_pedido,id_producto	id_pedido	8	tienda_online.p.id_pedido,const	1	100.00	Using index

Explain 4 (Conteo de Detalles de Pedido para Producto y Cliente Específicos): Altamente optimizada. Esta es la consulta modelo. Lo más importante es que utiliza índices de forma exhaustiva y eficiente (Using index) para filtrar y unir las tablas. Esto significa que MySQL obtiene toda la información necesaria directamente de los índices, evitando lecturas de tabla completas y operaciones costosas.

Propuestas de índices de mejora

Pedidos por Estado: Crea un índice en Pedidos(estado, id_cliente). Esto hará que las búsquedas de pedidos pendientes sean rápidas y ayudará a agrupar los resultados por cliente de forma más eficiente.
Productos por Stock: Añade un índice en Productos(stock). Así, encontrar productos con stock disponible o bajo será mucho más fácil.
Productos por Categoría: Tener un índice en Productos(id_categoria). Esto es clave para que las búsquedas de productos por su categoría sean muy rápidas.
Detalles de Pedido por Producto: Crea un índice en Detalles_Pedido(id_producto, id_pedido). Esto acelerará las consultas que necesiten verificar compras específicas de un producto.

Conclusiones Generales y lecciones aprendidas

Conclusión General

Para finalizar este proyecto de base de datos sobre la tienda online representa un sistema que cumple con las características esperadas y contiene una buena estructura gracias a crear un buen diseño y seguir las reglas especificadas hasta la tercera forma normal lo uqe garantía integridad, elimina redundancias y optimiza su funcionamiento en general, ademas mediante el uso de claves primarias, foráneas y restricciones adecuadas. Las tablas (Clientes, Productos, Categorías, Pedidos, Detalles_Pedido y Resenas) cubren las funcionalidades de una tienda en línea, como la gestión de inventario, pedidos, y reseñas de clientes.

Los scripts SQL relizados (Tablas.sql, Poblar.sql, Sentencias.sql, Procedimientos.sql) proporcionan una estructura fuerte para la creación, población, y operaciones dentro de la base de datos, por su parte los procedimientos automatizar aquellas tareas comunes que se pueden realizar y se hagan de c´manera consistente, ya para concluir este proyecto presenta una guía clara y

legible de lo que se a echo y los conocimientos ques e tiene en este proyecto aunque hayan existido algunas complicaciones.

Lecciones Aprendidas

1. **Importancia de la Normalización**:

   La normalización hasta 3NF fue una muy buena práctica para evitar redundancias y anomalías en las operaciones de inserción, actualización y eliminación. Aprendí que un diseño bien normalizado reduce el almacenamiento innecesario y facilita el mantenimiento, aunque requiere un análisis cuidadoso para identificar dependencias funcionales.

2. **Uso Estratégico de Restricciones**:
   La implementación de restricciones como UNIQUE, CHECK, y claves foráneas asegura una integridad referencial y previene errores comunes, como productos duplicados o pedidos con cantidades negativas. mas sin embargo no se debe abusar de estos recurso para no limitar demasiado la base de datos
3. **Ventajas de los Procedimientos Almacenados**:
   Los procedimientos almacenados simplifican operaciones complejas y centralizan la lógica. Esto reduce errores en aplicaciones cliente y mejora la consistencia.

4. **Optimización con Índices**:
   La creación de índices en Tablas.sql mejora el rendimiento de consultas frecuentes. Se aprendió que identificar las consultas más comunes al diseñar la base de datos es crucial para decidir qué índices implementar, equilibrando el rendimiento con el costo de mantenimiento de los índices.
5. **Pruebas Rigurosas**:
   Al probar los scripts y procedimientos, descubrimos la importancia de validar casos extremos (por ejemplo, pedidos con stock insuficiente o reseñas de clientes no autorizados). Esto nos enseñó a anticipar errores potenciales y a incluir manejos de excepciones (como SIGNAL SQLSTATE en los

6. **Herramientas Visuales**:
   El uso de PlantUML para el diagrama ER (Diagrma(ER).txt) fue una herramienta muy útil para visualizar las relaciones entre tablas.
7. **Organización del Proyecto**:
   La organización del proyecto y el uso de control de versiones (Git) facilitan la gestión del proyecto. Esto mostró la importancia de establecer una estructura clara desde el inicio para mejorar la accesibilidad y tener una mejor visualización de lo que se ha hecho a través del tiempo.

Referencias bibliográficas

MySQL :: MySQL 8.4 Reference Manual :: 13.3.5 The ENUM type. (s. f.). https://dev.mysql.com/doc/refman/8.4/en/enum.html

Monroy, G. (2025). *Consultas SQL básicas, procedimientos almacenados y triggers* [Notas de clase]. Licenciatura en Sistemas y Tecnologías de Información, Universidad Autónoma Metropolitana, Unidad Cuajimalpa.

Monroy, G. (2025). *Joins* [Notas de clase]. Licenciatura en Sistemas y Tecnologías de Información, Universidad Autónoma Metropolitana, Unidad Cuajimalpa.

Monroy, G. (2025). *Normalización* [Notas de clase]. Licenciatura en Sistemas y Tecnologías de Información, Universidad Autónoma Metropolitana, Unidad Cuajimalpa.