Proyecto Final Tienda Online de Productos Electrónicos

Alumno: **Aaron Rodrigo Ramos Reyes**

Profesor: **Guillermo Monroy Rodríguez**

Materia: **Bases de Datos**

Universidad Autónoma Metropolitana

Unidad Cuajimalpa

1 de agosto de 2025

Índice

1 Introducción

1.1 Introducción al Proyecto

El proyecto consiste en el diseño e implementación de una base de datos en **MySQL**, la cual debe cumplir con todos los requisitos descritos en el documento *"ProyectoFinalBD"*.

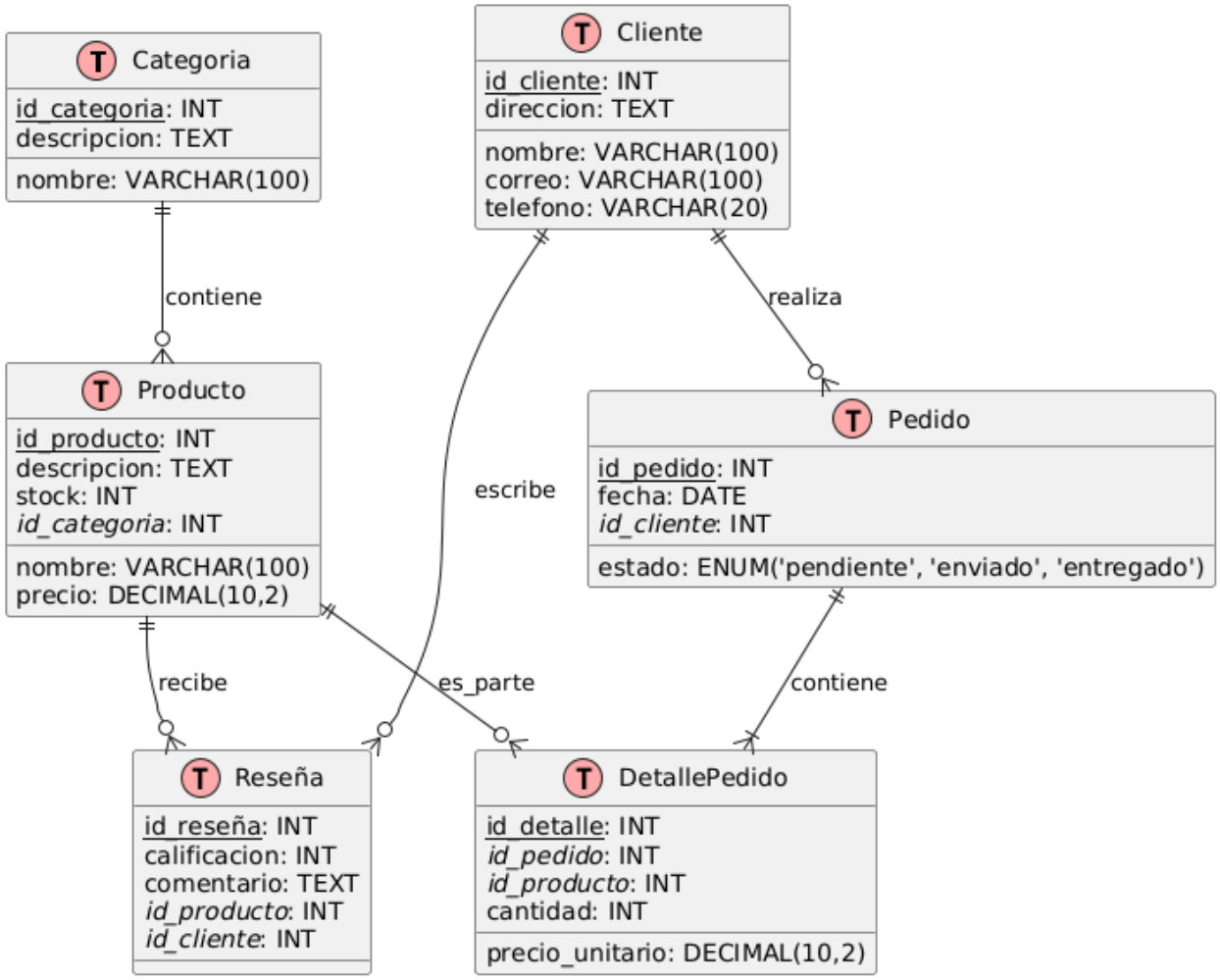
Con este propósito, se requerirá el uso y reforzamiento de tecnologías como **PlantUML** para la elaboración de diagramas, LaTeX para generar documentación clara, estructurada y modificable, y **MySQL** para la implementación de la base de datos. Asimismo, se utilizarán otras herramientas como los *stored procedures* para extender la funcionalidad de la base de datos.

Adicionalmente, se plantea la posibilidad de desarrollar un sistema en **Java** utilizando una arquitectura de tres capas, lo cual será opcional y dependerá del tiempo disponible para la entrega. Esta etapa adicional busca integrar y aplicar conocimientos adquiridos en otros cursos del plan de estudios.

2 Diagrama ER y justificación de la normalización

2.1 Diagrama Entidad-Relación

El siguiente diagrama muestra el modelo entidad-relación para el sistema de gestión de una tienda en línea de productos electrónicos. Se incluyen entidades como productos, clientes, pedidos, categorías, y reseñas, así como las relaciones entre ellas.



2.2 Justificación de la normalización

El modelo ha sido diseñado cumpliendo con las tres primeras formas normales para asegurar consistencia, evitar redundancia y facilitar el mantenimiento de los datos:

- **Primera Forma Normal (1FN):** Todas las columnas contienen valores atómicos y no hay grupos repetitivos.
- **Segunda Forma Normal (2FN):** No existen dependencias parciales; todos los atributos no clave dependen completamente de la clave primaria.
- **Tercera Forma Normal (3FN):** No hay dependencias transitivas; cada atributo no clave depende solamente de la clave primaria.

3 Implementación de tablas en MySQL

Este capitulo presenta la implementacion del modelo entidad-relacion en el sistema de gestion de bases de datos MySQL. Se creo una base de datos llamada `TiendaOnline` y a continuacion se describen las tablas que la conforman.

3.1 Tabla Categoria

La tabla `Categoria` organiza los productos en grupos tematicos como telefonos, laptops o accesorios. Contiene una clave primaria que la identifica de forma unica.

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

Descripcion de campos:

- `id_categoria`: Identificador unico de la categoria.
- `nombre`: Nombre de la categoria (por ejemplo, "Laptops").
- `descripcion`: Descripcion opcional de la categoria.

3.2 Tabla Producto

Almacena los productos disponibles en la tienda. Cada producto pertenece a una categoria.

```
CREATE TABLE Producto (
  id_producto INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  descripcion TEXT,
  precio DECIMAL(10,2) NOT NULL,
  stock INT NOT NULL CHECK (stock >= 0),
  id_categoria INT,
  FOREIGN KEY (id_categoria) REFERENCES Categoria(id_categoria)
);
```

Descripcion de campos:

- `id_producto`: Identificador del producto.
- `nombre`, `descripcion`, `precio`: Informacion basica.

- `stock`: Inventario disponible, no puede ser negativo.
- `id_categoria`: Clave foranea que enlaza con la tabla `Categoria`.

3.3 Tabla Cliente

Contiene la informacion personal de cada cliente. El correo electronico debe ser unico.

```
CREATE TABLE Cliente (
  id_cliente INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  correo VARCHAR(100) NOT NULL UNIQUE,
  direccion TEXT,
  telefono VARCHAR(20)
);
```

Descripcion de campos:

- `id_cliente`: Identificador del cliente.
- `correo`: Sirve como clave candidata por ser unico.
- `direccion`, `telefono`: Informacion de contacto.

3.4 Tabla Pedido

Almacena las ordenes hechas por los clientes, incluyendo la fecha y el estado del pedido.

```
CREATE TABLE Pedido (
  id_pedido INT AUTO_INCREMENT PRIMARY KEY,
  fecha DATE NOT NULL,
  estado ENUM('pendiente', 'enviado', 'entregado') NOT NULL,
  id_cliente INT,
  FOREIGN KEY (id_cliente) REFERENCES Cliente(id_cliente)
);
```

Descripcion de campos:

- `id_pedido`: Clave primaria del pedido.
- `fecha`: Fecha en que se realizo.
- `estado`: Estado actual del pedido (controlado por ENUM).
- `id_cliente`: Clave foranea hacia el cliente que realizo el pedido.

3.5 Tabla DetallePedido

Esta tabla representa una relacion `muchos a muchos` entre `Pedido` y `Producto`. Registra los productos incluidos en cada pedido, sus cantidades y precios al momento de la compra.

```
CREATE TABLE DetallePedido (
  id_detalle INT AUTO_INCREMENT PRIMARY KEY,
  id_pedido INT,
  id_producto INT,
  cantidad INT NOT NULL,
  precio_unitario DECIMAL(10,2) NOT NULL,
  FOREIGN KEY (id_pedido) REFERENCES Pedido(id_pedido),
  FOREIGN KEY (id_producto) REFERENCES Producto(id_producto)
);
```

Descripcion detallada:

- `id_detalle`: Identificador unico de cada linea del pedido.
- `id_pedido`: Clave foranea hacia el pedido correspondiente.
- `id_producto`: Clave foranea hacia el producto que se esta comprando.
- `cantidad`: Numero de unidades del producto en ese pedido.
- `precio_unitario`: Precio del producto en el momento del pedido (puede variar del precio actual).

Esta tabla es esencial para conservar un historico detallado de cada compra, permitiendo consultas como "¿que productos se compraron en el pedido 123?.º "¿cuantas unidades de producto A se vendieron en total?".

3.6 Tabla Resena

Permite registrar las calificaciones y comentarios hechos por los clientes sobre los productos comprados.

```
CREATE TABLE Resena (
  id_resena INT AUTO_INCREMENT PRIMARY KEY,
  calificacion INT CHECK (calificacion BETWEEN 1 AND 5),
  comentario TEXT,
  id_producto INT,
  id_cliente INT,
  FOREIGN KEY (id_producto) REFERENCES Producto(id_producto),
  FOREIGN KEY (id_cliente) REFERENCES Cliente(id_cliente)
);
```

Descripcion de campos:

- `id_resena`: Identificador de la resena.
- `calificacion`: Numero entero entre 1 y 5.
- `comentario`: Texto libre.
- `id_producto`, `id_cliente`: Claves foraneas para validar que el cliente resena un producto especifico.

4 Implementación de índices y triggers en MySQL

En esta sección se presentan los índices y triggers implementados para mejorar el rendimiento de consultas frecuentes y asegurar restricciones de negocio directamente en la base de datos.

4.1 Índices implementados

Se definieron los siguientes índices secundarios para optimizar el acceso a los datos mediante claves foráneas o combinaciones frecuentes de columnas.

```
CREATE INDEX idx_producto_categoria ON Producto(id_categoria);
CREATE INDEX idx_pedido_cliente ON Pedido(id_cliente);
CREATE INDEX idx_detalle_pedido_producto ON DetallePedido(id_producto);
CREATE INDEX idx_resena_cliente_producto ON Resena(id_cliente, id_producto
    );
```

Explicación de cada índice:

- `idx_producto_categoria` Mejora las consultas que recuperan productos por categoría, como: `SELECT * FROM Producto WHERE id_categoria = 3;`
- `idx_pedido_cliente` Optimiza las búsquedas de pedidos realizados por un cliente: `SELECT * FROM Pedido WHERE id_cliente = 12;`
- `idx_detalle_pedido_producto` Acelera la identificación de pedidos que incluyen un producto específico: `SELECT * FROM DetallePedido WHERE id_producto = 5;`
- `idx_resena_cliente_producto` Este índice compuesto facilita consultas como: `SELECT * FROM Reseña WHERE id_cliente = 1 AND id_producto = 10;` lo cual es útil para validar si un cliente ya reseñó un producto.

4.2 Trigger: Límite de 5 pedidos pendientes por cliente

Para restringir la cantidad de pedidos pendientes a un máximo de cinco por cliente, se implementó el siguiente trigger:

```
DELIMITER //

CREATE TRIGGER trg_max_pedidos_pendientes
BEFORE INSERT ON Pedido
FOR EACH ROW
BEGIN
  DECLARE pedidos_pendientes INT;

  SELECT COUNT(*) INTO pedidos_pendientes
  FROM Pedido
  WHERE id_cliente = NEW.id_cliente AND estado = 'pendiente';

  IF pedidos_pendientes >= 5 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'El cliente ya tiene 5 pedidos pendientes.';
  END IF;
END;
//

DELIMITER ;
```

Explicación:

- El trigger se ejecuta antes de insertar un nuevo pedido.
- Cuenta los pedidos del cliente con estado `pendiente`.
- Si el número es igual o mayor a 5, se lanza una excepción y se bloquea la inserción.
- Esto garantiza que se cumpla la restricción incluso fuera del sistema de interfaz (por ejemplo, por scripts o inyecciones directas).

4.3 Trigger: Validar reseñas solo de clientes que hayan comprado el producto

Este trigger impide que un cliente deje una reseña sobre un producto si no lo ha comprado previamente. Así se protege la integridad del sistema de calificaciones.

```
DELIMITER //

CREATE TRIGGER trg_resena_solo_si_compro
BEFORE INSERT ON Resena
FOR EACH ROW
BEGIN
  DECLARE total_compras INT;
```

```
8
9     SELECT COUNT(*) INTO total_compras
10    FROM Pedido P
11    JOIN DetallePedido D ON P.id_pedido = D.id_pedido
12    WHERE P.id_cliente = NEW.id_cliente
13      AND D.id_producto = NEW.id_producto;
14
15    IF total_compras = 0 THEN
16      SIGNAL SQLSTATE '45000'
17      SET MESSAGE_TEXT = 'El cliente no ha comprado este producto.';
18    END IF;
19  END;
20  //
21
22  DELIMITER ;
```

Explicación:

- El trigger se ejecuta antes de insertar una fila en la tabla `Reseña`.
- Se hace una consulta a las tablas `Pedido` y `DetallePedido` para verificar si el cliente efectivamente compró el producto.
- Si el resultado es cero, significa que el cliente no lo ha comprado.
- En ese caso, el trigger lanza un error personalizado que bloquea la inserción de la reseña.

Este control asegura que las reseñas provienen exclusivamente de clientes reales, reforzando la autenticidad de la retroalimentación.

5 Consultas de almacenamiento en MySQL

En esta sección se presentan tres procedimientos almacenados implementados en MySQL que permiten realizar consultas comunes dentro del sistema de gestión de una tienda en línea. Cada procedimiento tiene una finalidad específica y fue optimizado para consultar múltiples tablas relacionadas.

5.1 Productos disponibles por categoría, ordenados por precio

Este procedimiento lista todos los productos disponibles (con stock mayor a cero), agrupados por categoría y ordenados por precio en orden ascendente.

```
1  DELIMITER //
2
```

```
CREATE PROCEDURE ProductosPorCategoria()
BEGIN
  SELECT
    c.nombre AS categoria,
    p.nombre AS producto,
    p.descripcion,
    p.precio,
    p.stock
  FROM Producto p
  JOIN Categoria c ON p.id_categoria = c.id_categoria
  WHERE p.stock > 0
  ORDER BY c.nombre, p.precio ASC;
END;
//

DELIMITER ;
```

Explicación:

- Se realiza un unión (`JOIN`) entre la tabla `Producto` y `Categoria`.
- Solo se incluyen productos disponibles, es decir, con `stock >0`.
- El ordenamiento se realiza primero por nombre de categoría y luego por precio.

5.2 Clientes con pedidos pendientes y total de compras

Este procedimiento muestra los clientes que tienen pedidos en estado pendiente, junto con el total de pedidos que han realizado.

```
DELIMITER //

CREATE PROCEDURE PedidosPendientes()
BEGIN
  SELECT
    cl.id_cliente,
    cl.nombre,
    cl.correo,
    COUNT(CASE WHEN p.estado = 'pendiente' THEN 1 END) AS
       pedidos_pendientes,
    COUNT(p.id_pedido) AS total_pedidos
  FROM Cliente cl
  LEFT JOIN Pedido p ON cl.id_cliente = p.id_cliente
  GROUP BY cl.id_cliente, cl.nombre, cl.correo
  HAVING pedidos_pendientes > 0;
END;
//

DELIMITER ;
```

Explicación:

- Se realiza una `LEFT JOIN` entre clientes y sus pedidos.
- Se usa una condición `CASE WHEN` para contar únicamente los pedidos pendientes.
- Se utiliza `HAVING` para filtrar y mostrar solo los clientes con al menos un pedido pendiente.

5.3 Top 5 productos con mejor calificación promedio

Este procedimiento obtiene los cinco productos mejor calificados según la media de reseñas dadas por los clientes.

```
DELIMITER //

CREATE PROCEDURE TopProductos()
BEGIN
  SELECT
    p.nombre AS producto,
    ROUND(AVG(r.calificacion), 2) AS promedio_calificacion,
    COUNT(r.id_resena) AS total_resenas
  FROM Producto p
  JOIN Resena r ON p.id_producto = r.id_producto
  GROUP BY p.id_producto, p.nombre
  HAVING COUNT(r.id_resena) > 0
  ORDER BY promedio_calificacion DESC
  LIMIT 5;
END;
//

DELIMITER;
```

Explicación:

- Se calcula el promedio de calificaciones para cada producto.
- Se filtran productos que no tienen reseñas.
- Se ordena el resultado en orden descendente por calificación y se limita a los primeros cinco.

6 Procedimientos almacenados en MySQL

En esta sección se presentan los procedimientos almacenados implementados para gestionar operaciones fundamentales del sistema. Se incluyen:

1. **Registrar un nuevo pedido:** Verifica el límite de 5 pedidos pendientes por cliente y que haya stock suficiente. *Nota:* En este procedimiento se actualiza el stock de los productos, por lo que no es necesario un procedimiento adicional para la actualización del stock.
2. **Registrar una reseña:** Permite insertar una reseña, delegando la validación a un trigger ya creado. *Nota:* La validación se realiza en el trigger que se encuentra en la sección **Implementación de índices y triggers en MySQL, Sub sección: "Trigger: validar reseñas solo de clientes que hayan comprado el producto"**.
3. **Cambiar el estado de un pedido:** Permite actualizar de forma segura el estado de un pedido (por ejemplo, de pendiente a enviado).

6.1 1. Registrar un nuevo pedido

El siguiente procedimiento almacena un nuevo pedido, verificando que el cliente no tenga ya 5 pedidos pendientes y que exista stock suficiente para cada producto incluido. Se utiliza un parámetro JSON para enviar los detalles del pedido (lista de productos y cantidades).

```
DELIMITER //

CREATE PROCEDURE RegistrarPedido(
  IN p_id_cliente INT,
  IN p_fecha DATE,
  IN p_detalles JSON -- Ej: [{"id_producto":1, "cantidad":2}, {"
      id_producto":3, "cantidad":1}]
)
BEGIN
  DECLARE pedidos_pend INT;
  DECLARE stock_actual INT;
  DECLARE i INT DEFAULT 0;
  DECLARE detalles_count INT;
  DECLARE id_pedido_nuevo INT;
  DECLARE v_id_producto INT;
  DECLARE v_cantidad INT;

  -- Verificar que el cliente tenga menos de 5 pedidos pendientes
  SELECT COUNT(*) INTO pedidos_pend
  FROM Pedido
  WHERE id_cliente = p_id_cliente AND estado = 'pendiente';

  IF pedidos_pend >= 5 THEN
    SIGNAL SQLSTATE '45000'
```

```
    SET MESSAGE_TEXT = 'El cliente ya tiene 5 pedidos pendientes.';
  END IF;

  -- Crear el pedido
  INSERT INTO Pedido(fecha, estado, id_cliente)
  VALUES (p_fecha, 'pendiente', p_id_cliente);

  SET id_pedido_nuevo = LAST_INSERT_ID();
  SET detalles_count = JSON_LENGTH(p_detalles);

  -- Procesar cada detalle del pedido
  WHILE i < detalles_count DO
    SET v_id_producto = JSON_UNQUOTE(JSON_EXTRACT(p_detalles, CONCAT('$[',
        i, '].id_producto')));
    SET v_cantidad = JSON_UNQUOTE(JSON_EXTRACT(p_detalles, CONCAT('$[', i,
        '].cantidad')));

    -- Verificar stock disponible
    SELECT stock INTO stock_actual FROM Producto WHERE id_producto =
       v_id_producto;
    IF stock_actual < v_cantidad THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = CONCAT('Stock insuficiente para el producto ID: '
         , v_id_producto);
    END IF;

    -- Insertar detalle del pedido
    INSERT INTO DetallePedido(id_pedido, id_producto, cantidad,
       precio_unitario)
    SELECT id_pedido_nuevo, v_id_producto, v_cantidad, precio
    FROM Producto
    WHERE id_producto = v_id_producto;

    -- Actualizar el stock (actualizacion incluida en este procedimiento)
    UPDATE Producto
    SET stock = stock - v_cantidad
    WHERE id_producto = v_id_producto;

    SET i = i + 1;
  END WHILE;
END;
//

DELIMITER ;
```

6.2 2. Registrar una resena

Este procedimiento inserta una resena para un producto. La validacion de que el cliente haya comprado el producto se realiza mediante un trigger (consulta la seccion Implementacion de indices y triggers en MySQL).

```
DELIMITER //

CREATE PROCEDURE RegistrarResena(
  IN p_id_cliente INT,
  IN p_id_producto INT,
  IN p_calificacion INT,
  IN p_comentario TEXT
)
BEGIN

INSERT INTO Resena(calificacion, comentario, id_producto, id_cliente)
VALUES (p_calificacion, p_comentario, p_id_producto, p_id_cliente);
END;
//

DELIMITER ;
```

Nota: El trigger utilizado para validar que el cliente haya comprado el producto se encuentra en la sección **Implementación de índices y triggers en MySQL**, en la subsección *"Trigger: validar reseñas solo de clientes que hayan comprado el producto"*.

6.3 3. Cambiar el estado de un pedido

Este procedimiento permite actualizar el estado de un pedido existente de forma segura.

```
DELIMITER //

CREATE PROCEDURE CambiarEstadoPedido(
  IN p_id_pedido INT,
  IN p_nuevo_estado ENUM('pendiente', 'enviado', 'entregado')
)
BEGIN
  DECLARE pedido_existente INT;

  -- Verificar que el pedido existe
  SELECT COUNT(*) INTO pedido_existente
  FROM Pedido
  WHERE id_pedido = p_id_pedido;

  IF pedido_existente = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'El pedido no existe.';
  END IF;

  -- Actualizar el estado del pedido
  UPDATE Pedido
  SET estado = p_nuevo_estado
  WHERE id_pedido = p_id_pedido;
END;
//
```

```
26
27 DELIMITER ;
```

Explicación general:

- En el procedimiento **RegistrarPedido** se verifica previamente que el cliente no exceda el límite de 5 pedidos pendientes y que cada producto cuente con stock suficiente. Se procesan los detalles del pedido enviados en formato JSON, y se actualiza el stock de cada producto al momento de generar el pedido.
- En el procedimiento **RegistrarResena**, la validación que asegura que el cliente haya comprado el producto se delega al trigger implementado, eliminando así redundancias en la lógica.
- En el procedimiento **CambiarEstadoPedido**, se valida la existencia del pedido antes de actualizar su estado, garantizando la integridad de la operación.