

# 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

2 de agosto de 2025

# Índice

<b>1. Introducción</b>	<b>3</b>
1.1. Introducción al Proyecto . . . . .	3
<b>2. Diagrama ER y justificación de la normalización</b>	<b>3</b>
2.1. Diagrama Entidad-Relación . . . . .	3
2.2. Justificación de la normalización . . . . .	4
<b>3. Implementación de tablas en MySQL</b>	<b>5</b>
3.1. Tabla Categoría . . . . .	5
3.2. Tabla Producto . . . . .	5
3.3. Tabla Cliente . . . . .	6
3.4. Tabla Pedido . . . . .	6
3.5. Tabla DetallePedido . . . . .	7
3.6. Tabla Resena . . . . .	7
<b>4. Implementación de índices y triggers en MySQL</b>	<b>8</b>
4.1. Índices implementados . . . . .	8
4.2. Trigger: Límite de 5 pedidos pendientes por cliente . . . . .	9
4.3. Trigger: Validar reseñas solo de clientes que hayan comprado el producto . . . . .	9
<b>5. Consultas de almacenamiento en MySQL</b>	<b>10</b>
5.1. Productos disponibles por categoría, ordenados por precio . . . . .	10
5.2. Clientes con pedidos pendientes y total de compras . . . . .	11
5.3. Top 5 productos con mejor calificación promedio . . . . .	12
<b>6. Procedimientos almacenados en MySQL</b>	<b>13</b>

6.1.	1. Registrar un nuevo pedido . . . . .	13
6.2.	2. Registrar una resena . . . . .	14
6.3.	3. Cambiar el estado de un pedido . . . . .	15
<b>7.</b>	<b>Procedimientos almacenados (Parte II)</b>	<b>16</b>
7.1.	Eliminar reseñas de un producto y actualizar promedio . . . . .	16
7.2.	Agregar producto evitando duplicados por nombre y categoría . . . . .	17
7.3.	Actualizar direccion y telefono de un cliente . . . . .	17
7.4.	Reporte de productos con bajo stock . . . . .	18
<b>8.</b>	<b>Ejemplos de uso de índices</b>	<b>18</b>
8.1.	Índice idx_producto_categoria . . . . .	19
8.2.	Índice idx_pedido_cliente . . . . .	19
8.3.	Índice idx_detalle_pedido_producto . . . . .	19
8.4.	Índice idx_resena_cliente_producto . . . . .	19
8.5.	Resumen . . . . .	20

# 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*".

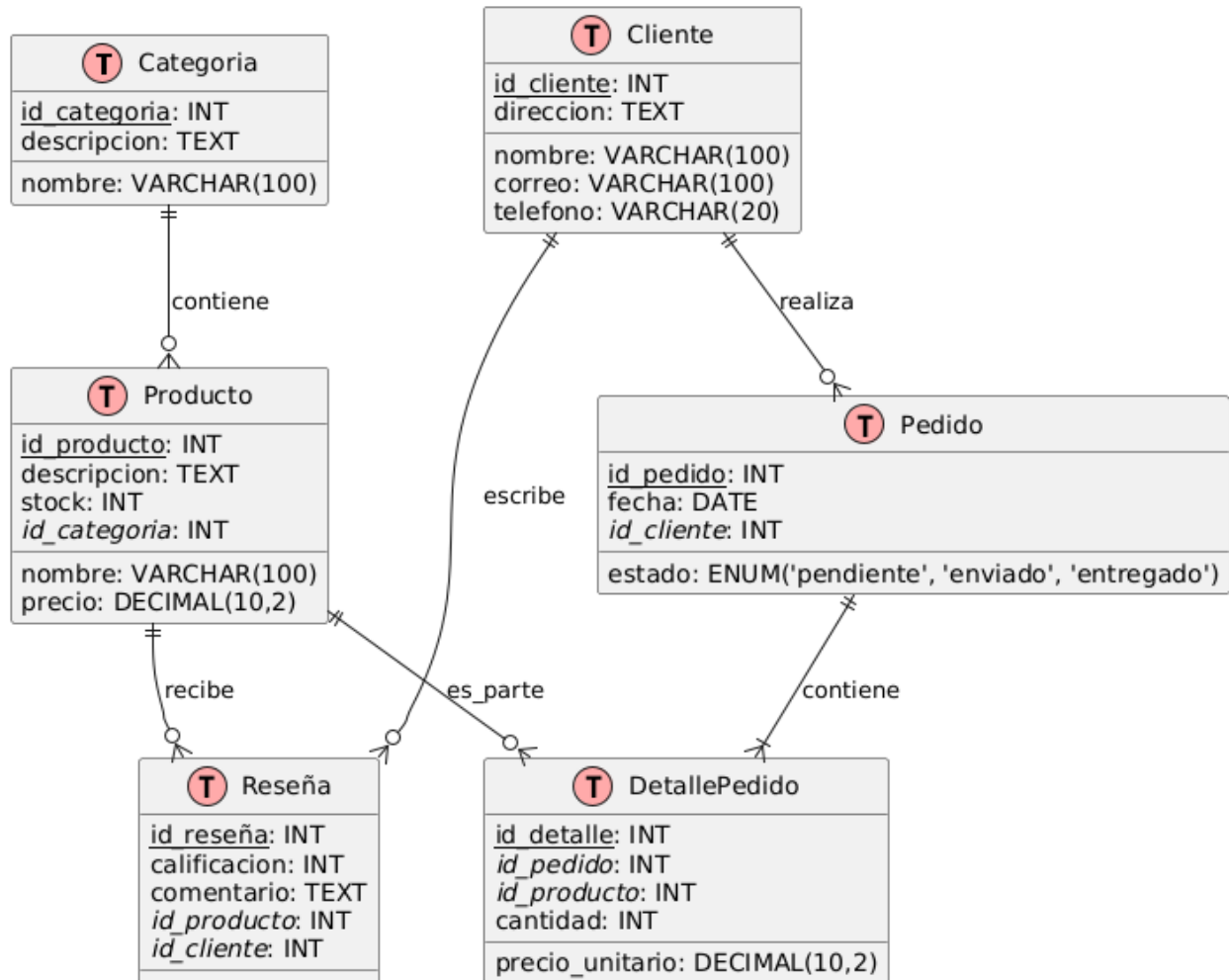
Con este propósito, se requerirá el uso y reforzamiento de tecnologías como **PlantUML** para la elaboración de diagramas, **L<sup>A</sup>T<sub>E</sub>X** para generar documentación clara, estructurada y modificable, y **MySQL** para la implementación de la base de datos. Asimismo, se utilizarán otras 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 capítulo presenta la implementación del modelo entidad-relación en el sistema de gestión de bases de datos MySQL. Se creó una base de datos llamada **TiendaOnline** y a continuación se describen las tablas que la conforman.

### 3.1 Tabla Categoría

La tabla **Categoría** organiza los productos en grupos temáticos como teléfonos, laptops o accesorios. Contiene una clave primaria que la identifica de forma única.

```
1 CREATE TABLE Categoría (  
2     id_categoria INT AUTO_INCREMENT PRIMARY KEY,  
3     nombre VARCHAR(100) NOT NULL,  
4     descripcion TEXT  
5 );
```

**Descripción de campos:**

- **id\_categoria:** Identificador único de la categoría.
- **nombre:** Nombre de la categoría (por ejemplo, "Laptops").
- **descripcion:** Descripción opcional de la categoría.

### 3.2 Tabla Producto

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

```
1 CREATE TABLE Producto (  
2     id_producto INT AUTO_INCREMENT PRIMARY KEY,  
3     nombre VARCHAR(100) NOT NULL,  
4     descripcion TEXT,  
5     precio DECIMAL(10,2) NOT NULL,  
6     stock INT NOT NULL CHECK (stock >= 0),  
7     id_categoria INT,  
8     FOREIGN KEY (id_categoria) REFERENCES Categoría(id_categoria)  
9 );
```

**Descripción de campos:**

- **id\_producto:** Identificador del producto.
- **nombre, descripcion, precio:** Información básica.

- 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.

```

1 CREATE TABLE Cliente (
2   id_cliente INT AUTO_INCREMENT PRIMARY KEY,
3   nombre VARCHAR(100) NOT NULL,
4   correo VARCHAR(100) NOT NULL UNIQUE,
5   direccion TEXT,
6   telefono VARCHAR(20)
7 );

```

#### 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.

```

1 CREATE TABLE Pedido (
2   id_pedido INT AUTO_INCREMENT PRIMARY KEY,
3   fecha DATE NOT NULL,
4   estado ENUM('pendiente', 'enviado', 'entregado') NOT NULL,
5   id_cliente INT,
6   FOREIGN KEY (id_cliente) REFERENCES Cliente(id_cliente)
7 );

```

#### 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.

```
1 CREATE TABLE DetallePedido (  
2   id_detalle INT AUTO_INCREMENT PRIMARY KEY,  
3   id_pedido INT,  
4   id_producto INT,  
5   cantidad INT NOT NULL,  
6   precio_unitario DECIMAL(10,2) NOT NULL,  
7   FOREIGN KEY (id_pedido) REFERENCES Pedido(id_pedido),  
8   FOREIGN KEY (id_producto) REFERENCES Producto(id_producto)  
9 );
```

#### 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?." "¿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.

```
1 CREATE TABLE Resena (  
2   id_resena INT AUTO_INCREMENT PRIMARY KEY,  
3   calificacion INT CHECK (calificacion BETWEEN 1 AND 5),  
4   comentario TEXT,  
5   id_producto INT,  
6   id_cliente INT,  
7   FOREIGN KEY (id_producto) REFERENCES Producto(id_producto),  
8   FOREIGN KEY (id_cliente) REFERENCES Cliente(id_cliente)  
9 );
```



### Descripción 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.

```
1 CREATE INDEX idx_producto_categoria ON Producto(id_categoria);
2 CREATE INDEX idx_pedido_cliente ON Pedido(id_cliente);
3 CREATE INDEX idx_detalle_pedido_producto ON DetallePedido(id_producto);
4 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 Resena 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:

```
1 DELIMITER //
```

```
2
```

```
3 CREATE TRIGGER trg_max_pedidos_pendientes
```

```
4 BEFORE INSERT ON Pedido
```

```
5 FOR EACH ROW
```

```
6 BEGIN
```

```
7     DECLARE pedidos_pendientes INT;
```

```
8
```

```
9     SELECT COUNT(*) INTO pedidos_pendientes
```

```
10    FROM Pedido
```

```
11    WHERE id_cliente = NEW.id_cliente AND estado = 'pendiente';
```

```
12
```

```
13    IF pedidos_pendientes >= 5 THEN
```

```
14        SIGNAL SQLSTATE '45000'
```

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

```
16    END IF;
```

```
17 END;
```

```
18 //
```

```
19
```

```
20 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.

```
1 DELIMITER //
```

```
2
```

```
3 CREATE TRIGGER trg_resena_solo_si_compro
```

```
4 BEFORE INSERT ON Resena
```

```
5 FOR EACH ROW
```

```
6 BEGIN
```

```
7     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

```

```

3 CREATE PROCEDURE ProductosPorCategoria()
4 BEGIN
5     SELECT
6         c.nombre AS categoria,
7         p.nombre AS producto,
8         p.descripcion,
9         p.precio,
10        p.stock
11    FROM Producto p
12    JOIN Categoria c ON p.id_categoria = c.id_categoria
13    WHERE p.stock > 0
14    ORDER BY c.nombre, p.precio ASC;
15 END;
16 //
17
18 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.

```

1 DELIMITER //
2
3 CREATE PROCEDURE PedidosPendientes()
4 BEGIN
5     SELECT
6         cl.id_cliente,
7         cl.nombre,
8         cl.correo,
9         COUNT(CASE WHEN p.estado = 'pendiente' THEN 1 END) AS
10        pedidos_pendientes,
11        COUNT(p.id_pedido) AS total_pedidos
12    FROM Cliente cl
13    LEFT JOIN Pedido p ON cl.id_cliente = p.id_cliente
14    GROUP BY cl.id_cliente, cl.nombre, cl.correo
15    HAVING pedidos_pendientes > 0;
16 END;
17 //
18 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.

```
1 DELIMITER //
```

```
2
```

```
3 CREATE PROCEDURE TopProductos()
```

```
4 BEGIN
```

```
5     SELECT
```

```
6         p.nombre AS producto,
```

```
7         ROUND(AVG(r.calificacion), 2) AS promedio_calificacion,
```

```
8         COUNT(r.id_resena) AS total_resenas
```

```
9     FROM Producto p
```

```
10    JOIN Resena r ON p.id_producto = r.id_producto
```

```
11    GROUP BY p.id_producto, p.nombre
```

```
12    HAVING COUNT(r.id_resena) > 0
```

```
13    ORDER BY promedio_calificacion DESC
```

```
14    LIMIT 5;
```

```
15 END;
```

```
16 //
```

```
17
```

```
18 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).

```
1 DELIMITER //
2
3 CREATE PROCEDURE RegistrarPedido(
4     IN p_id_cliente INT,
5     IN p_fecha DATE,
6     IN p_detalle JSON -- Ej: [{"id_producto":1, "cantidad":2}, {"
7         id_producto":3, "cantidad":1}]
8 )
9 BEGIN
10     DECLARE pedidos_pend INT;
11     DECLARE stock_actual INT;
12     DECLARE i INT DEFAULT 0;
13     DECLARE detalles_count INT;
14     DECLARE id_pedido_nuevo INT;
15     DECLARE v_id_producto INT;
16     DECLARE v_cantidad INT;
17
18     -- Verificar que el cliente tenga menos de 5 pedidos pendientes
19     SELECT COUNT(*) INTO pedidos_pend
20     FROM Pedido
21     WHERE id_cliente = p_id_cliente AND estado = 'pendiente';
22
23     IF pedidos_pend >= 5 THEN
24         SIGNAL SQLSTATE '45000';
```

```

24     SET MESSAGE_TEXT = 'El cliente ya tiene 5 pedidos pendientes.';
25 END IF;
26
27 -- Crear el pedido
28 INSERT INTO Pedido(fecha, estado, id_cliente)
29 VALUES (p_fecha, 'pendiente', p_id_cliente);
30
31 SET id_pedido_nuevo = LAST_INSERT_ID();
32 SET detalles_count = JSON_LENGTH(p_detalle);
33
34 -- Procesar cada detalle del pedido
35 WHILE i < detalles_count DO
36     SET v_id_producto = JSON_UNQUOTE(JSON_EXTRACT(p_detalle, CONCAT('$[',
37     i, '].id_producto')));
38     SET v_cantidad = JSON_UNQUOTE(JSON_EXTRACT(p_detalle, CONCAT('$[', i,
39     '].cantidad')));
40
41     -- Verificar stock disponible
42     SELECT stock INTO stock_actual FROM Producto WHERE id_producto =
43     v_id_producto;
44     IF stock_actual < v_cantidad THEN
45         SIGNAL SQLSTATE '45000'
46         SET MESSAGE_TEXT = CONCAT('Stock insuficiente para el producto ID:',
47         , v_id_producto);
48     END IF;
49
50     -- Insertar detalle del pedido
51     INSERT INTO DetallePedido(id_pedido, id_producto, cantidad,
52     precio_unitario)
53     SELECT id_pedido_nuevo, v_id_producto, v_cantidad, precio
54     FROM Producto
55     WHERE id_producto = v_id_producto;
56
57     -- Actualizar el stock (actualizacion incluida en este procedimiento)
58     UPDATE Producto
59     SET stock = stock - v_cantidad
60     WHERE id_producto = v_id_producto;
61
62     SET i = i + 1;
63 END WHILE;
64 END;
65 //
66 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).

```

1 DELIMITER //
2
3 CREATE PROCEDURE RegistrarResena(
4     IN p_id_cliente INT,
5     IN p_id_producto INT,
6     IN p_calificacion INT,
7     IN p_comentario TEXT
8 )
9 BEGIN
10
11 INSERT INTO Resena(calificacion, comentario, id_producto, id_cliente)
12 VALUES (p_calificacion, p_comentario, p_id_producto, p_id_cliente);
13 END;
14 //
15
16 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.

```

1 DELIMITER //
2
3 CREATE PROCEDURE CambiarEstadoPedido(
4     IN p_id_pedido INT,
5     IN p_nuevo_estado ENUM('pendiente', 'enviado', 'entregado')
6 )
7 BEGIN
8     DECLARE pedido_existente INT;
9
10     -- Verificar que el pedido existe
11     SELECT COUNT(*) INTO pedido_existente
12     FROM Pedido
13     WHERE id_pedido = p_id_pedido;
14
15     IF pedido_existente = 0 THEN
16         SIGNAL SQLSTATE '45000'
17         SET MESSAGE_TEXT = 'El pedido no existe.';
18     END IF;
19
20     -- Actualizar el estado del pedido
21     UPDATE Pedido
22     SET estado = p_nuevo_estado
23     WHERE id_pedido = p_id_pedido;
24 END;
25 //

```



```
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.

## 7 Procedimientos almacenados (Parte II)

Esta sección presenta una segunda tanda de procedimientos almacenados que permiten realizar operaciones comunes en la tienda en línea, desde mantenimiento de productos hasta reportes y validaciones adicionales.

### 7.1 Eliminar reseñas de un producto y actualizar promedio

```
1 DELIMITER //  
2  
3 CREATE PROCEDURE EliminarResenasYActualizarPromedio(  
4     IN p_id_producto INT  
5 )  
6 BEGIN  
7     DELETE FROM Resena  
8     WHERE id_producto = p_id_producto;  
9 END;  
10 //  
11  
12 DELIMITER ;
```

**Explicación:** Este procedimiento elimina todas las reseñas asociadas a un producto y, si se lleva el promedio en una columna, se puede actualizar a NULL para indicar que ya no tiene calificaciones.

## 7.2 Agregar producto evitando duplicados por nombre y categoría

```
1 DELIMITER //
```

```
2
```

```
3 CREATE PROCEDURE AgregarProductoSinDuplicado(
```

```
4     IN p_nombre VARCHAR(100),
```

```
5     IN p_descripcion TEXT,
```

```
6     IN p_precio DECIMAL(10,2),
```

```
7     IN p_stock INT,
```

```
8     IN p_id_categoria INT
```

```
9 )
```

```
10 BEGIN
```

```
11     DECLARE existe INT;
```

```
12
```

```
13     SELECT COUNT(*) INTO existe
```

```
14     FROM Producto
```

```
15     WHERE nombre = p_nombre AND id_categoria = p_id_categoria;
```

```
16
```

```
17     IF existe > 0 THEN
```

```
18         SIGNAL SQLSTATE '45000'
```

```
19         SET MESSAGE_TEXT = 'Ya existe un producto con ese nombre en esta
```

```
20         categor\u00eda.';
```

```
21     ELSE
```

```
22         INSERT INTO Producto(nombre, descripcion, precio, stock, id_categoria)
```

```
23         VALUES (p_nombre, p_descripcion, p_precio, p_stock, p_id_categoria);
```

```
24     END IF;
```

```
25 END;
```

```
26 //
```

```
27 DELIMITER ;
```

**Explicación:** Asegura que no se repitan productos con el mismo nombre dentro de una misma categoría, arrojando un error si ya existe.

## 7.3 Actualizar direccion y telefono de un cliente

```
1 DELIMITER //
```

```
2
```

```
3 CREATE PROCEDURE ActualizarDatosCliente(
```

```
4     IN p_id_cliente INT,
```

```
5     IN p_nueva_direccion TEXT,
```

```
6     IN p_nuevo_telefono VARCHAR(20)
```

```
7 )
```

```
8 BEGIN
```

```
9     DECLARE existe INT;
```

```
10
```

```
11     SELECT COUNT(*) INTO existe
```

```
12     FROM Cliente
```

```
13     WHERE id_cliente = p_id_cliente;
```

```
14
```

```
15     IF existe = 0 THEN
```

```

16     SIGNAL SQLSTATE '45000'
17     SET MESSAGE_TEXT = 'El_cliente_no_existe.';
18 ELSE
19     UPDATE Cliente
20     SET direccion = p_nueva_direccion,
21         telefono = p_nuevo_telefono
22     WHERE id_cliente = p_id_cliente;
23 END IF;
24 END;
25 //
26
27 DELIMITER ;

```

**Explicación:** Permite modificar de forma segura los datos de contacto de un cliente, asegurando que el cliente exista antes de aplicar los cambios.

## 7.4 Reporte de productos con bajo stock

```

1 DELIMITER //
2
3 CREATE PROCEDURE ReporteStockBajo()
4 BEGIN
5     SELECT id_producto, nombre, stock
6     FROM Producto
7     WHERE stock < 5
8     ORDER BY stock ASC;
9 END;
10 //
11
12 DELIMITER ;

```

**Explicación:** Devuelve un listado de productos con stock inferior a 5 unidades, ordenado de menor a mayor.

Estos procedimientos mejoran la robustez del sistema al controlar integridad de datos, simplificar operaciones y facilitar consultas comunes para mantenimiento y gestión.

## 8 Ejemplos de uso de índices

Para mejorar el rendimiento de las consultas en la base de datos, se definieron varios índices sobre columnas que participan frecuentemente en búsquedas, relaciones o agrupamientos. A continuación, se detallan los índices creados, los stored procedures donde se utilizan y la parte del código donde actúan.

### 8.1 Índice idx\_producto\_categoria

```
1 CREATE INDEX idx_producto_categoria ON Producto(id_categoria);
```

Este índice se usa en el stored procedure `ProductosPorCategoria` (sección de consultas almacenadas) para acelerar la cláusula `WHERE id_categoria = p_categoria`. Parte del código:

```
1 WHERE p.id_categoria = c.id_categoria
2     AND p.stock > 0
```

### 8.2 Índice idx\_pedido\_cliente

```
1 CREATE INDEX idx_pedido_cliente ON Pedido(id_cliente);
```

Este índice mejora la performance en el procedimiento `RegistrarPedido` (sección de procedimientos operativos) cuando verifica el número de pedidos pendientes:

```
1 SELECT COUNT(*) INTO pedidos_pend
2 FROM Pedido
3 WHERE id_cliente = p_id_cliente
4     AND estado = 'pendiente';
```

### 8.3 Índice idx\_detalle\_pedido\_producto

```
1 CREATE INDEX idx_detalle_pedido_producto ON DetallePedido(id_producto);
```

Este índice acelera la inserción de detalles en `RegistrarPedido`, concretamente en la validación de stock y la inserción en `DetallePedido`:

```
1 SELECT stock INTO stock_actual
2 FROM Producto
3 WHERE id_producto = v_id_producto;
```

### 8.4 Índice idx\_resena\_cliente\_producto

```
1 CREATE INDEX idx_resena_cliente_producto ON Resena(id_cliente, id_producto
2 );
```

Se utiliza en el trigger `trg_resena_solo_si_compro` (sección ??) para validar rápidamente que el cliente compró el producto:

```
1 SELECT COUNT(*) INTO total_compras
2 FROM Pedido P
3 JOIN DetallePedido D ON P.id_pedido = D.id_pedido
4 WHERE P.id_cliente = NEW.id_cliente
5      AND D.id_producto = NEW.id_producto;
```

## 8.5 Resumen

Los índices descritos se integran en los stored procedures y triggers principales para:

- Reducir tiempos de respuesta en consultas de productos por categoría.
- Acelerar validaciones en RegistrarPedido y trg\_resena\_solo\_si\_compro.
- Mejorar la eficiencia en operaciones críticas de negocio.