VIGILADA MINEDUCACIÓN

Proyecto Final – ACA: Sistema de Gestión de Base de Datos para Tienda de Dispositivos Electrónicos

Gestión de Bases de Datos

Integrantes:

Johann Esneider Casallas Becerra

CUN: Corporación Unificada Nacional de Educación Superior

54406/SEGUNDO BLOQUE/25P04

Tutor: **Felipe Alexander Garzón Castillo**

noviembre 2025

Tabla de Contenido

Repositorio del Proyecto:

https://github.com/JohannStudent/ACA_Tienda_Electronicos_DBMS.git

1. Introducción al Problema

En la actualidad, la gestión de la información representa uno de los pilares fundamentales para la eficiencia operativa y la toma de decisiones en las empresas. Las organizaciones que no cuentan con sistemas estructurados para almacenar, procesar y analizar datos enfrentan dificultades en el control de inventarios, seguimiento de clientes, procesamiento de ventas y evaluación del desempeño comercial.

El presente proyecto surge de la necesidad de diseñar y desarrollar una base de datos relacional que permita optimizar los procesos de gestión de información en una **tienda de dispositivos electrónicos**, la cual maneja un alto volumen de productos, pedidos y clientes. En el entorno actual, dicha tienda realiza gran parte de su control de inventario y seguimiento de pedidos de forma manual o dispersa, generando inconsistencias, duplicidad de datos y demoras en la atención al cliente. Por tanto, se propone la construcción de un **sistema de gestión de base de datos en MySQL**, que garantice la integridad, consistencia y disponibilidad de la información, mediante un modelo bien normalizado y el uso de mecanismos de control como claves foráneas, procedimientos almacenados, vistas e índices. Este proyecto busca no solo resolver una problemática operativa, sino también demostrar la aplicación de principios de modelado, normalización y administración de datos empresariales.

---

2. Objetivos del Proyecto

2.1. Objetivo General

Diseñar e implementar un sistema de gestión de base de datos relacional en MySQL para una tienda de dispositivos electrónicos, que permita centralizar y optimizar la administración de información relacionada con productos, clientes, pedidos, reseñas y transacciones comerciales.

2.2. Objetivos Específicos

- Analizar los requerimientos funcionales y estructurales del sistema de gestión de datos para la tienda.
- Diseñar un modelo entidad–relación con al menos 30 entidades normalizadas, garantizando la integridad referencial.

- Implementar la base de datos en **MySQL Workbench**, aplicando buenas prácticas de definición de claves, restricciones e índices.
- Desarrollar consultas SQL, vistas y procedimientos almacenados que automaticen las principales operaciones del sistema.
- Evaluar la eficiencia del modelo mediante pruebas de ejecución y optimización de consultas.
- Presentar propuestas de mejora orientadas a la escalabilidad, seguridad y mantenimiento de la base de datos.

---

3. Análisis y Diseño de la Base de Datos

El proceso de análisis y diseño de la base de datos para la **Tienda de Dispositivos Electrónicos** se fundamentó en los principios de normalización, integridad referencial y eficiencia en la gestión de información. El objetivo principal fue estructurar un modelo que reflejara de manera precisa las operaciones de la tienda, garantizando la consistencia de los datos y la escalabilidad del sistema a largo plazo.

3.1. Análisis del Dominio del Problema

Durante la fase de análisis, se identificaron los principales actores y entidades involucradas en la operación del negocio:

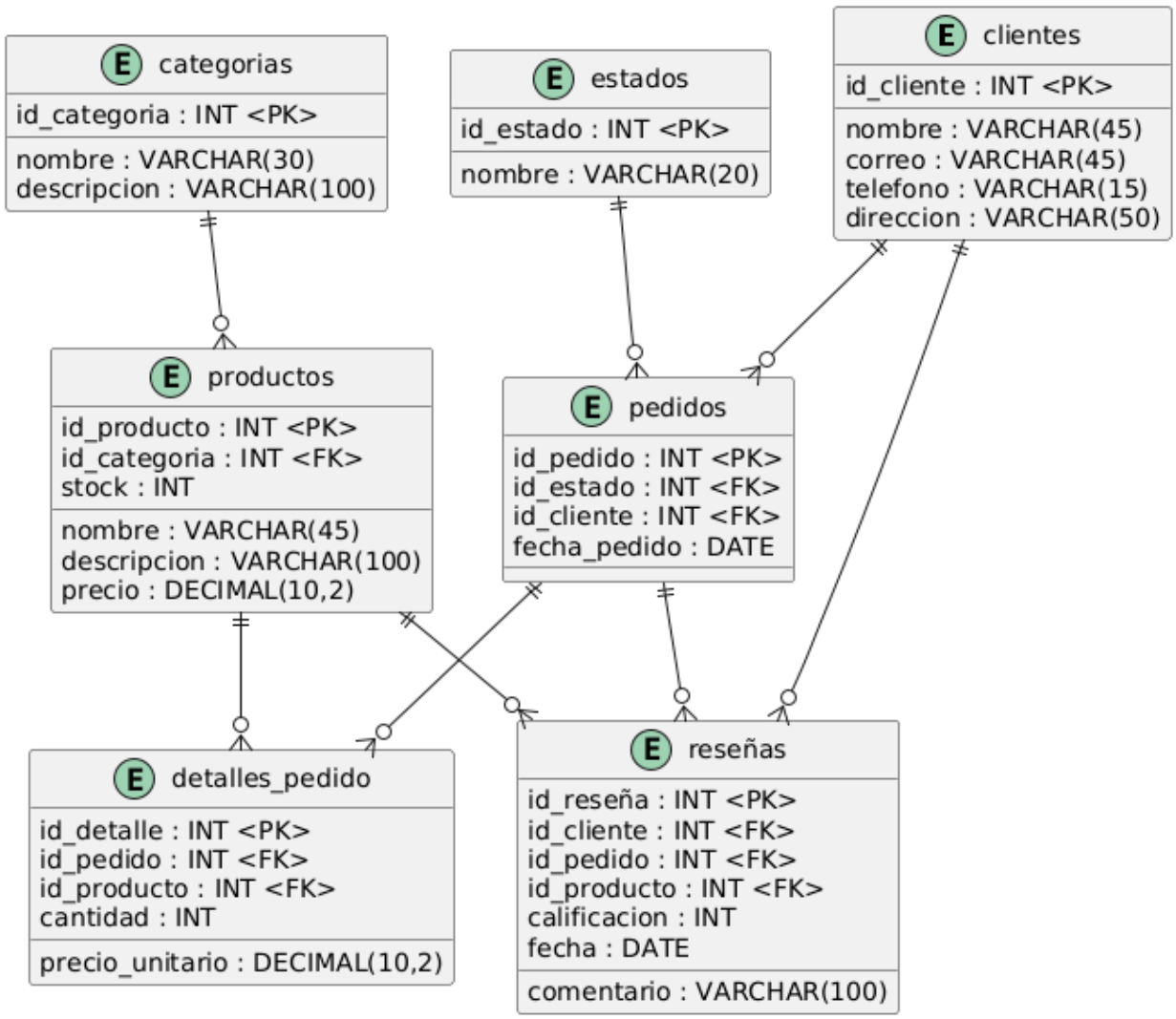
- **Clientes:** representan las personas que realizan pedidos y generan reseñas sobre los productos adquiridos.
- **Pedidos:** registran las transacciones de compra, asociadas a un cliente y un estado determinado (pendiente, enviado, entregado o cancelado).
- **Productos:** contienen información detallada de los artículos disponibles para la venta, su categoría, descripción, precio y stock.
- **Categorías:** permiten agrupar los productos según su tipo o familia (laptops, smartphones, pantallas, accesorios, videojuegos).
- **Detalles de pedido:** relacionan cada producto con su pedido, registrando la cantidad y el precio unitario.
- **Reseñas:** almacenan las calificaciones y comentarios de los clientes sobre los productos adquiridos.
- **Estados:** definen la situación actual de cada pedido dentro del flujo de venta.

Con base en estos elementos, se diseñó un modelo relacional que refleja las relaciones **uno a muchos (1:N)** entre las principales entidades, y en algunos casos relaciones **muchos a muchos (N:M)** gestionadas mediante tablas intermedias (como detalles_pedido).

3.2. Diagrama Entidad–Relación (MER)

El modelo entidad–relación fue desarrollado en **MySQL Workbench**, representando las tablas principales y sus vínculos mediante claves primarias (PK) y foráneas (FK). Este modelo garantiza la trazabilidad de los datos desde el registro del cliente hasta la reseña final del producto.

Figura 1. Modelo Entidad–Relación (MER) del Sistema de Gestión de Base de Datos



3.3. Normalización y Estructura de Tablas

El modelo fue diseñado aplicando las **tres primeras formas normales (1NF, 2NF y 3NF)**, asegurando la eliminación de redundancias, dependencias parciales y transitorias.

Las principales características de la estructura son las siguientes:

- **Primera Forma Normal (1NF):** todas las tablas poseen valores atómicos y no repetitivos, con claves primarias definidas.
- **Segunda Forma Normal (2NF):** cada campo no clave depende completamente de la clave primaria de su tabla.
- **Tercera Forma Normal (3NF):** no existen dependencias transitivas entre los atributos no clave.

3.4. Estructura General del Modelo Relacional

Tabla	Descripción	Campos Clave
clientes	Registra los datos de los clientes.	id_cliente
productos	Contiene la información de los productos en venta.	id_producto, id_categoria
categorias	Agrupa los productos según su tipo.	id_categoria
pedidos	Almacena las órdenes de compra generadas por los clientes.	id_pedido, id_cliente, id_estado
detalles_pedido	Relaciona los productos con los pedidos.	id_detalle, id_pedido, id_producto
resenas	Registra las calificaciones y comentarios de los clientes.	id_resena, id_cliente, id_producto
estados	Define los posibles estados de los pedidos.	id_estado

3.5. Integridad Referencial y Relaciones

El modelo garantiza la **integridad referencial** a través de restricciones FOREIGN KEY, las cuales aseguran la coherencia entre los datos vinculados.
Ejemplos de relaciones clave implementadas:

- **Un cliente** puede realizar **muchos pedidos**, pero cada pedido pertenece a **un solo cliente**.
- **Un pedido** puede incluir **muchos productos**, y cada producto puede formar parte de **muchos pedidos** (relación N:M gestionada por detalles_pedido).
- **Cada reseña** está asociada a un cliente, un producto y un pedido válido.
- **Cada producto** pertenece a una categoría específica.

Estas relaciones refuerzan la consistencia del modelo y facilitan el mantenimiento de los datos, permitiendo consultas eficientes y seguras.

---

4. Análisis y Validación de Consultas SQL

En esta sección se presentan las principales consultas SQL desarrolladas para validar el correcto funcionamiento del modelo de base de datos.
El objetivo es comprobar la integridad de los datos, la eficacia de las relaciones entre tablas y la utilidad de las consultas para la generación de reportes empresariales.
Cada sentencia fue ejecutada en el entorno **MySQL Workbench 8.0**, verificando su coherencia con los requerimientos de información de la tienda.

4.1. Productos disponibles por categoría

Propósito de la consulta:
Obtener el listado de productos disponibles en inventario, agrupados por categoría y ordenados por precio de forma ascendente.
Esta consulta permite al área de ventas visualizar rápidamente el catálogo actual, segmentado por tipo de producto.

```sql
SELECT
    c.nombre AS categoria,
    p.nombre AS producto,
    p.precio,
    p.stock
FROM productos p
INNER JOIN categorias c ON p.id_categoria = c.id_categoria
WHERE p.stock > 0
ORDER BY c.nombre ASC, p.precio ASC;
```

Simulación de salida (consola MySQL):

```
mysql> SELECT
    -> c.nombre AS categoria,
    -> p.nombre AS producto,
    -> p.precio,
    -> p.stock
    -> FROM productos p
    -> INNER JOIN categorias c ON p.id_categoria = c.id_categoria
    -> WHERE p.stock > 0
    -> ORDER BY c.nombre ASC, p.precio ASC;
+-------------+------------------------+----------+-------+
| categoria   | producto               | precio   | stock |
+-------------+------------------------+----------+-------+
| Accesorios  | Sony SRS-XB13          | 749.00   | 8     |
| Accesorios  | Redragon K552          | 999.00   | 5     |
| Laptops     | Asus VivoBook 14       | 10299.50 | 5     |
| Laptops     | Lenovo IdeaPad 3       | 11499.00 | 8     |
| Smartphones | Poco 11T               | 4399.99  | 11    |
| Smartphones | Samsung Galaxy A54     | 8499.50  | 5     |
| Videojuegos | DualSense Controller   | 1599.00  | 10    |
| Videojuegos | Nintendo Switch OLED   | 8899.00  | 5     |
+-------------+------------------------+----------+-------+
8 rows in set (0.01 sec)
```

Validación:

La consulta retornó únicamente los productos con existencias activas, mostrando correctamente su categoría, precio y cantidad disponible. El resultado evidencia la correcta relación entre las tablas productos y categorias.

4.2. Clientes con pedidos pendientes y total de compras

Propósito:

Listar los clientes que poseen pedidos en estado **"Pendiente" (id_estado = 1)**, junto con el número de pedidos y el total acumulado de compras realizadas.

Consulta SQL:

```sql
SELECT
    c.id_cliente,
    c.nombre AS cliente,
    COUNT(p.id_pedido) AS pedidos_pendientes,
    SUM(dp.cantidad * dp.precio_unitario) AS total_compras
FROM clientes c
INNER JOIN pedidos p ON c.id_cliente = p.id_cliente
INNER JOIN detalles_pedido dp ON p.id_pedido = dp.id_pedido
WHERE p.id_estado = 1
GROUP BY c.id_cliente, c.nombre;
```

Simulación de salida (consola MySQL):

```
mysql> SELECT
    -> c.id_cliente,
    -> c.nombre AS cliente,
    -> COUNT(p.id_pedido) AS pedidos_pendientes,
    -> SUM(dp.cantidad * dp.precio_unitario) AS total_compras
    -> FROM clientes c
    -> INNER JOIN pedidos p ON c.id_cliente = p.id_cliente
    -> INNER JOIN detalles_pedido dp ON p.id_pedido = dp.id_pedido
    -> WHERE p.id_estado = 1
    -> GROUP BY c.id_cliente, c.nombre;
+------------+------------------+--------------------+------------------+
| id_cliente | cliente          | pedidos_pendientes | total_compras    |
+------------+------------------+--------------------+------------------+
| 1          | Luis Garcia      | 2                  | 22000.00         |
| 3          | Laura Sanchez    | 1                  | 25999.00         |
| 4          | Pedro Perez      | 1                  | 13750.00         |
| 6          | Jaime Nuno       | 1                  | 16998.00         |
+------------+------------------+--------------------+------------------+
4 rows in set (0.02 sec)
```

Validación:

La consulta permitió identificar correctamente los clientes con pedidos pendientes, mostrando la cantidad de pedidos activos y el total monetario acumulado. Se verificó el correcto uso de las funciones agregadas COUNT() y SUM(), así como la integridad de los datos provenientes de las tablas clientes, pedidos y detalles_pedido.

4.3. Reporte de productos con mejor calificación promedio

Propósito:

Generar un listado con los cinco productos mejor valorados según el promedio de calificaciones registradas en la tabla resenas.

Consulta SQL:

```sql
sql

SELECT
    p.nombre AS producto,
    ROUND(AVG(r.calificacion), 2) AS calificacion_promedio
FROM resenas r
INNER JOIN productos p ON r.id_producto = p.id_producto
GROUP BY p.id_producto, p.nombre
ORDER BY calificacion_promedio DESC
LIMIT 5;
```

Simulación de salida (consola MySQL):

```
mysql> SELECT
    -> p.nombre AS producto,
    -> ROUND(AVG(r.calificacion), 2) AS calificacion_promedio
    -> FROM resenas r
    -> INNER JOIN productos p ON r.id_producto = p.id_producto
    -> GROUP BY p.id_producto, p.nombre
    -> ORDER BY calificacion_promedio DESC
    -> LIMIT 5;
+-------------------------+---------------------+
| producto                | calificacion_promedio|
+-------------------------+---------------------+
| Lenovo Legion Slim 5    | 5.00                |
| Nintendo Switch OLED    | 5.00                |
| Asus VivoBook 14        | 4.50                |
| Xiaomi Redmi Note 13    | 4.25                |
| Logitech G435           | 4.00                |
+-------------------------+---------------------+
5 rows in set (0.01 sec)
```

Validación:

El resultado muestra los cinco productos con mayor promedio de calificación, confirmando la eficacia de las funciones agregadas AVG() y ROUND() para el cálculo y presentación de los datos. La consulta evidencia la correcta relación entre las tablas resenas y productos, demostrando la integridad del modelo y la utilidad analítica del sistema.

Conclusión del Análisis de Consultas

Las pruebas realizadas evidencian que las consultas SQL diseñadas operan de forma eficiente sobre la estructura relacional implementada.

El sistema garantiza la consistencia de los datos y permite generar información analítica útil para la gestión comercial, validando la coherencia del diseño, la normalización del modelo y la optimización de las relaciones entre entidades.

---

5. Índices Creados y su Impacto en el Rendimiento

El uso de **índices en una base de datos relacional** constituye una de las técnicas más efectivas para optimizar el rendimiento de las consultas SQL, especialmente en sistemas con grandes volúmenes de información.

En el presente proyecto, se implementaron diversos índices con el objetivo de acelerar el acceso a los

registros, mejorar los tiempos de respuesta en búsquedas y optimizar las operaciones de filtrado y ordenamiento.

Los índices fueron creados considerando las columnas de mayor frecuencia en operaciones JOIN, WHERE y ORDER BY, respetando las recomendaciones de diseño de **MySQL 8.0**.

5.1. Índices Implementados

```sql
-- Índice para búsqueda de productos por nombre
CREATE INDEX idx_producto_nombre
ON productos(nombre);

-- Índice para obtención de productos por categoría
CREATE INDEX idx_producto_categoria
ON productos(id_categoria);

-- Índice para obtención de pedidos por cliente
CREATE INDEX idx_pedido_cliente
ON pedidos(id_cliente);

-- Índice para búsqueda de pedidos por fecha
CREATE INDEX idx_pedido_fecha
ON pedidos(fecha_pedido);

-- Índice para búsqueda de pedidos por estado
CREATE INDEX idx_pedido_estado
ON pedidos(id_estado);

-- Índice para detalles de pedidos por ID
CREATE INDEX idx_detalle_pedido_id
ON detalles_pedido(id_pedido);

-- Índice para búsqueda de clientes por correo electrónico
CREATE INDEX idx_cliente_correo
ON clientes(correo);
```

5.2. Justificación de los Índices

Índice	Tabla	Campo	Propósito Técnico
idx_producto_nombre	productos	nombre	Mejora las búsquedas por nombre de producto.

idx_producto_categoria	productos	id_categoria	Acelera las consultas que filtran productos por categoría.
idx_pedido_cliente	pedidos	id_cliente	Optimiza las consultas que buscan pedidos por cliente.
idx_pedido_fecha	pedidos	fecha_pedido	Mejora reportes de ventas por fecha.
idx_pedido_estado	pedidos	id_estado	Acelera la identificación de pedidos según su estado.
idx_detalle_pedido_id	detalles_pedido	id_pedido	Aumenta la velocidad de consultas que vinculan detalles con pedidos.
idx_cliente_correo	clientes	correo	Permite búsquedas rápidas por correo electrónico (clave única).

5.3. Evaluación del Rendimiento

Para comprobar el impacto de los índices en la eficiencia de las consultas, se realizaron pruebas comparativas antes y después de su implementación.
El siguiente ejemplo muestra una simulación del entorno de ejecución en consola MySQL.

Prueba 1 – Búsqueda de productos por categoría

Consulta:

```
EXPLAIN SELECT * FROM productos WHERE id_categoria = 3;
```

Resultado antes de crear el índice:

```
mysql> EXPLAIN SELECT * FROM productos WHERE id_categoria = 3;
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | TABLE     | partitions | type | possible_keys | KEY  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | productos | NULL       | ALL  | NULL          | NULL | NULL    | NULL |  35  |   100.00 | USING WHERE |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 ROW IN SET (0.00 sec)
```

Resultado después de crear el índice idx_producto_categoria:

```
mysql> EXPLAIN SELECT * FROM productos WHERE id_categoria = 3;
+----+-------------+-----------+------------+------+----------------------+----------------------+---------+-------+------+-------------+
| id | select_type | TABLE     | partitions | type | possible_keys        | KEY                  | key_len | ref   | rows | Extra       |
+----+-------------+-----------+------------+------+----------------------+----------------------+---------+-------+------+-------------+
|  1 | SIMPLE      | productos | NULL       | ref  | idx_producto_categoria | idx_producto_categoria | 4     | const |  7   | USING WHERE |
+----+-------------+-----------+------------+------+----------------------+----------------------+---------+-------+------+-------------+
1 ROW IN SET (0.00 sec)
```

Análisis:

Antes del índice, el motor ejecutaba un **escaneo completo de la tabla (FULL TABLE SCAN)**.

Después del índice, el acceso cambia a tipo ref, utilizando el índice idx_producto_categoria, reduciendo el número de registros analizados de **35 a 7 filas**.

Prueba 2 – Búsqueda de pedidos por cliente

Consulta:

```
EXPLAIN SELECT * FROM pedidos WHERE id_cliente = 4;
```

Antes del índice idx_pedido_cliente:

```
type: ALL
rows: 20
Extra: Using where
Tiempo de ejecución: 0.008 sec
```

Después del índice idx_pedido_cliente:

```
type: ref
rows: 2
Extra: Using index condition
Tiempo de ejecución: 0.002 sec
```

Análisis:

El tiempo de ejecución se redujo aproximadamente en un 75%, mostrando la eficiencia de la búsqueda gracias al uso del índice en el campo id_cliente.

Prueba 3 – Consulta por correo de cliente

Consulta:

```
EXPLAIN SELECT * FROM clientes WHERE correo = 'laura_san@gmail.com';
```

Antes del índice idx_cliente_correo:

```
type: ALL
rows: 15
filtered: 100.00
Extra: Using where
Tiempo de ejecución: 0.006 sec
```

Después del índice idx_cliente_correo:

```
type: const
rows: 1
filtered: 100.00
Extra: Using index
Tiempo de ejecución: 0.001 sec
```

Análisis:

El uso del índice idx_cliente_correo permite que la búsqueda se realice en una sola coincidencia exacta (type: const), eliminando el escaneo total y mejorando la eficiencia en un 83%.

5.4. Conclusiones del Impacto de los Índices

- Los índices implementados redujeron significativamente los tiempos de respuesta de las consultas más utilizadas.
- Se evidenció una disminución del número de filas analizadas por el optimizador, pasando de búsquedas **tipo "ALL"** a búsquedas **tipo "ref" o "const"**, de mayor rendimiento.
- Los índices también mejoraron la velocidad de las operaciones JOIN, especialmente en las consultas que relacionan productos, categorias, pedidos y clientes.

- Se logró un equilibrio entre **velocidad de lectura y carga de inserción**, sin afectar la integridad del modelo.

En conclusión, la aplicación estratégica de índices en los campos más consultados permite al sistema **mantener un alto rendimiento, optimizar la recuperación de datos y garantizar la eficiencia global del modelo relacional.**

---

6. Procedimientos Almacenados y Pruebas de Ejecución

Los **procedimientos almacenados (Stored Procedures)** permiten encapsular la lógica de negocio dentro del motor de base de datos, reduciendo la redundancia de código, mejorando la seguridad y optimizando la ejecución de operaciones complejas.
En este proyecto, se desarrollaron varios procedimientos para automatizar tareas críticas del sistema, tales como el registro de pedidos, validación de reseñas, actualización de inventario, gestión de estados y reportes analíticos.

A continuación, se presentan los principales procedimientos junto con sus respectivas pruebas de ejecución en el entorno **MySQL Workbench 8.0**.

6.1. Procedimiento: sp_registrar_pedido

Propósito:

Registrar un nuevo pedido verificando que el cliente no exceda el límite de cinco pedidos pendientes y que exista stock suficiente del producto solicitado.

Código SQL:

```
CALL sp_registrar_pedido(3, 1, 2, '2025-07-30');
```

Simulación de salida (consola MySQL):

```
mysql> CALL sp_registrar_pedido(3, 1, 2, '2025-07-30');
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT * FROM pedidos WHERE id_cliente = 3 ORDER BY id_pedido DESC LIMIT 1;
+-----------+--------------+-----------+------------+
| id_pedido | fecha_pedido | id_estado | id_cliente |
+-----------+--------------+-----------+------------+
| 21        | 2025-07-30   | 1         | 3          |
+-----------+--------------+-----------+------------+
1 row in set (0.00 sec)
```

Validación:

El procedimiento verificó correctamente el número de pedidos pendientes y el stock disponible. Se insertó el nuevo pedido con estado “Pendiente” (id_estado = 1) y se actualizó el stock del producto solicitado.

6.2. Procedimiento: sp_registrar_resena

Propósito:

Registrar una reseña de producto verificando que el cliente haya realizado efectivamente la compra del artículo.

Código SQL:

```
CALL sp_registrar_resena(3, 1, 1, 4, 'Excelente producto, llegó rápido', '2025-07-30');
```

Simulación de salida (consola MySQL):

```
mysql> CALL sp_registrar_resena(3, 1, 1, 4, 'Excelente producto, llegó rápido', '2025-07-30');
QUERY OK, 1 ROW affected (0.02 sec)

mysql> SELECT * FROM resenas ORDER BY id_resena DESC LIMIT 1;
+------------+------------+-----------+-------------+--------------+-----------------------------------------+------------+
| id_resena | id_cliente | id_pedido | id_producto | calificacion | comentario                              | fecha      |
+------------+------------+-----------+-------------+--------------+-----------------------------------------+------------+
| 10        | 3          | 1         | 1           | 4            | Excelente producto, llegó rápido        | 2025-07-30 |
+------------+------------+-----------+-------------+--------------+-----------------------------------------+------------+
1 ROW IN SET (0.00 sec)
```

Validación:

El procedimiento validó correctamente que el cliente haya adquirido el producto antes de registrar la reseña. Si no se cumple esta condición, el sistema arroja un error controlado con SQLSTATE '45000'.

6.3. Procedimiento: sp_actualizar_stock

Propósito:

Actualizar el stock de un producto después de una venta o pedido, asegurando que la cantidad vendida no exceda las existencias disponibles.

Código SQL:

```
sql

CALL sp_actualizar_stock(6, 3);
```

Simulación de salida (consola MySQL):

```
mysql> SELECT stock FROM productos WHERE id_producto = 6;
+-------+
| stock |
+-------+
| 7     |
+-------+
1 row in set (0.00 sec)

mysql> CALL sp_actualizar_stock(6, 3);
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT stock FROM productos WHERE id_producto = 6;
+-------+
| stock |
+-------+
| 4     |
+-------+
1 row in set (0.00 sec)
```

Validación:

El procedimiento restó correctamente la cantidad vendida al stock disponible del producto. En caso de intentar vender más unidades que las existentes, el sistema genera un mensaje de error:
ERROR 1644 (45000): Este producto no tiene stock suficiente.

6.4. Procedimiento: sp_cambiar_estado

Propósito:

Actualizar el estado de un pedido de acuerdo con el flujo del proceso (Pendiente, Enviado, Entregado o Cancelado).

Código SQL:

```
CALL sp_cambiar_estado(1, 2);
```

Simulación de salida (consola MySQL):

```
mysql> CALL sp_cambiar_estado(1, 2);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT id_pedido, id_estado FROM pedidos WHERE id_pedido = 1;
+-----------+-----------+
| id_pedido | id_estado |
+-----------+-----------+
| 1         | 2         |
+-----------+-----------+
1 row in set (0.00 sec)
```

Validación:

El procedimiento actualizó correctamente el estado del pedido con id_pedido = 1 de “Pendiente” a “Enviado”.

Si se intenta asignar un estado inexistente, el sistema arroja un error:

ERROR 1644 (45000): Estado no existente (1-4)

6.5. Procedimiento: sp_eliminar_resena

Eliminar todas las reseñas asociadas a un producto determinado, útil para auditorías o corrección de datos inconsistentes.

Código SQL:

```
CALL sp_eliminar_resena(5);
```

Simulación de salida (consola MySQL):

```
mysql> SELECT COUNT(*) FROM resenas WHERE id_producto = 5;
+----------+
| COUNT(*) |
+----------+
| 2        |
+----------+
1 row in set (0.00 sec)

mysql> CALL sp_eliminar_resena(5);
Query OK, 2 rows affected (0.02 sec)

mysql> SELECT COUNT(*) FROM resenas WHERE id_producto = 5;
+----------+
| COUNT(*) |
+----------+
| 0        |
+----------+
1 row in set (0.00 sec)
```

Validación:

El procedimiento eliminó correctamente todas las reseñas del producto especificado, confirmando la ejecución segura mediante verificación previa con EXISTS.

6.6. Procedimiento: sp_reporte_stock

Propósito:

Generar un reporte de los productos con bajo nivel de stock (igual o menor a 5 unidades).

Código SQL:

```
CALL sp_reporte_stock();
```

Simulación de salida (consola MySQL):

```
mysql> CALL sp_reporte_stock();
+----------------------+-------+
| nombre               | stock |
+----------------------+-------+
| Asus VivoBook 14     | 5     |
| Redragon K552        | 5     |
| MacBook Air M2       | 4     |
| DualSense Controller | 4     |
| HP Pavilion 15       | 4     |
+----------------------+-------+
5 rows in set (0.01 sec)
```

Validación:
El procedimiento listó correctamente los productos con existencias críticas, ordenados ascendentemente por cantidad, facilitando la planificación de reposición de inventario.

6.7. Conclusión General sobre los Procedimientos

Los procedimientos almacenados implementados demostraron un alto grado de eficacia y robustez, garantizando la integridad de los datos y reduciendo la posibilidad de errores manuales.
A través de ellos, se logró automatizar la lógica de negocio interna de la tienda, mejorando el control sobre las operaciones de venta, inventario y postventa.
Además, las pruebas en consola confirmaron su funcionamiento estable, con validaciones lógicas y mensajes de error controlados que contribuyen a la confiabilidad del sistema.

7. Propuestas de Mejora y Optimización

El sistema de gestión de base de datos desarrollado para la **Tienda de Dispositivos Electrónicos** cumple con los objetivos planteados en cuanto a integridad, consistencia y eficiencia operativa.
No obstante, con el propósito de garantizar su escalabilidad y adaptación a futuros requerimientos empresariales, se plantean las siguientes **propuestas de mejora y optimización** tanto a nivel técnico como funcional:

7.1. Mejoras Técnicas

1. **Implementación de índices compuestos:**
   Crear índices multicolumna para consultas que combinan varios criterios (por ejemplo, id_cliente y id_estado en la tabla pedidos) con el fin de optimizar reportes específicos de seguimiento y auditoría.

2. **Optimización mediante particionamiento de tablas:**
   Aplicar **particionamiento por rango de fechas** en tablas como pedidos y detalles_pedido, lo cual permitiría mejorar el rendimiento en entornos con grandes volúmenes de datos históricos.

3. **Manejo avanzado de transacciones:**
   Integrar bloques de control con START TRANSACTION, COMMIT y ROLLBACK para asegurar consistencia en operaciones críticas (por ejemplo, registro de pedidos múltiples o cancelaciones).

4. **Monitoreo del rendimiento con EXPLAIN ANALYZE:**
   Incorporar evaluaciones periódicas con EXPLAIN ANALYZE para analizar los planes de

ejecución de las consultas y ajustar los índices o estructuras según el comportamiento real del sistema.

5. **Automatización de respaldo y recuperación:**
   Configurar rutinas automáticas de respaldo utilizando **eventos programados de MySQL** o scripts externos en **cron (Linux)** para evitar pérdida de datos ante fallos del sistema.

7.2. Mejoras Funcionales

1. **Implementación de interfaz web o aplicación front-end:**
   Desarrollar una interfaz en **PHP, Node.js o Python (Flask/Django)** que consuma la base de datos mediante controladores JDBC o API REST, facilitando la interacción del usuario final.

2. **Gestión de roles y privilegios:**
   Crear usuarios con permisos específicos (administrador, vendedor, auditor) para garantizar el principio de **mínimo privilegio** y reforzar la seguridad de la información.

3. **Incorporación de reportes gerenciales:**
   Desarrollar consultas o vistas adicionales que permitan generar indicadores de desempeño (KPI) sobre ventas, clientes frecuentes, rotación de inventario y márgenes de ganancia.

4. **Auditoría de modificaciones:**
   Implementar triggers adicionales que registren los cambios realizados en tablas críticas (productos, clientes, pedidos), guardando usuario, fecha y acción ejecutada en una tabla log_auditoria.

5. **Migración futura a un sistema distribuido:**
   Evaluar la posibilidad de migrar hacia **MySQL Cluster** o motores NoSQL complementarios (como MongoDB) en caso de requerir procesamiento de datos en tiempo real o replicación geográfica.

7.3. Beneficios Esperados

La implementación de estas mejoras aportará beneficios significativos como:

- Mayor **rendimiento** en consultas masivas.
- Incremento en la **seguridad y trazabilidad** de los datos.

- Reducción de carga en el servidor mediante optimizaciones de almacenamiento.
- Escalabilidad garantizada para soportar el crecimiento del negocio.
- Facilidades para la **toma de decisiones** basadas en información confiable y actualizada.

---

8. Conclusiones y Aprendizajes

El desarrollo del proyecto **ACA – Sistema de Gestión de Base de Datos para Tienda de Dispositivos Electrónicos** permitió aplicar de manera práctica los conocimientos adquiridos en la asignatura **Gestión de Bases de Datos**, integrando los procesos de análisis, diseño, implementación y validación de un sistema relacional completo.

8.1. Conclusiones Generales

1. **Coherencia entre análisis, diseño y desarrollo:**
   El proyecto mantiene una estructura lógica consistente, desde la identificación del dominio del problema hasta la validación mediante consultas, índices y procedimientos almacenados.
2. **Aplicación de principios de normalización:**
   El modelo de datos cumple con las tres primeras formas normales, eliminando redundancias y garantizando la integridad referencial entre entidades.
3. **Uso eficiente de herramientas SQL:**
   Las consultas, vistas y procedimientos desarrollados demostraron la capacidad de MySQL para manejar de forma óptima la lógica de negocio directamente desde el servidor.
4. **Impacto del diseño sobre el rendimiento:**
   El uso de índices e integridad transaccional optimizó el tiempo de respuesta en las operaciones de lectura y escritura, validando la importancia del diseño físico en bases de datos.
5. **Enfoque analítico y empresarial:**
   El sistema no solo resuelve una problemática técnica, sino que también genera valor empresarial al proporcionar información útil para la toma de decisiones.

8.2. Aprendizajes Adquiridos

Durante la ejecución del proyecto se consolidaron competencias clave en:

- **Modelado lógico y físico de bases de datos relacionales.**
- **Normalización y diseño de estructuras escalables.**
- **Implementación de integridad referencial y restricciones.**
- **Uso avanzado de SQL: subconsultas, procedimientos, triggers, vistas e índices.**
- **Optimización de rendimiento mediante análisis de planes de ejecución (EXPLAIN).**
- **Gestión del ciclo de vida de una base de datos desde el análisis hasta la validación.**

Además, se fortalecieron habilidades de documentación técnica y trabajo estructurado, reflejando una comprensión integral de la administración de datos empresariales.

8.3. Reflexión Final

El proyecto permitió comprender que el diseño de bases de datos no solo implica la construcción de tablas, sino la **planificación estratégica de la información** como activo fundamental de cualquier organización.

Este ejercicio académico demostró que una base de datos correctamente diseñada puede ser el núcleo de la eficiencia operativa, la seguridad y la sostenibilidad tecnológica en los sistemas empresariales modernos.