

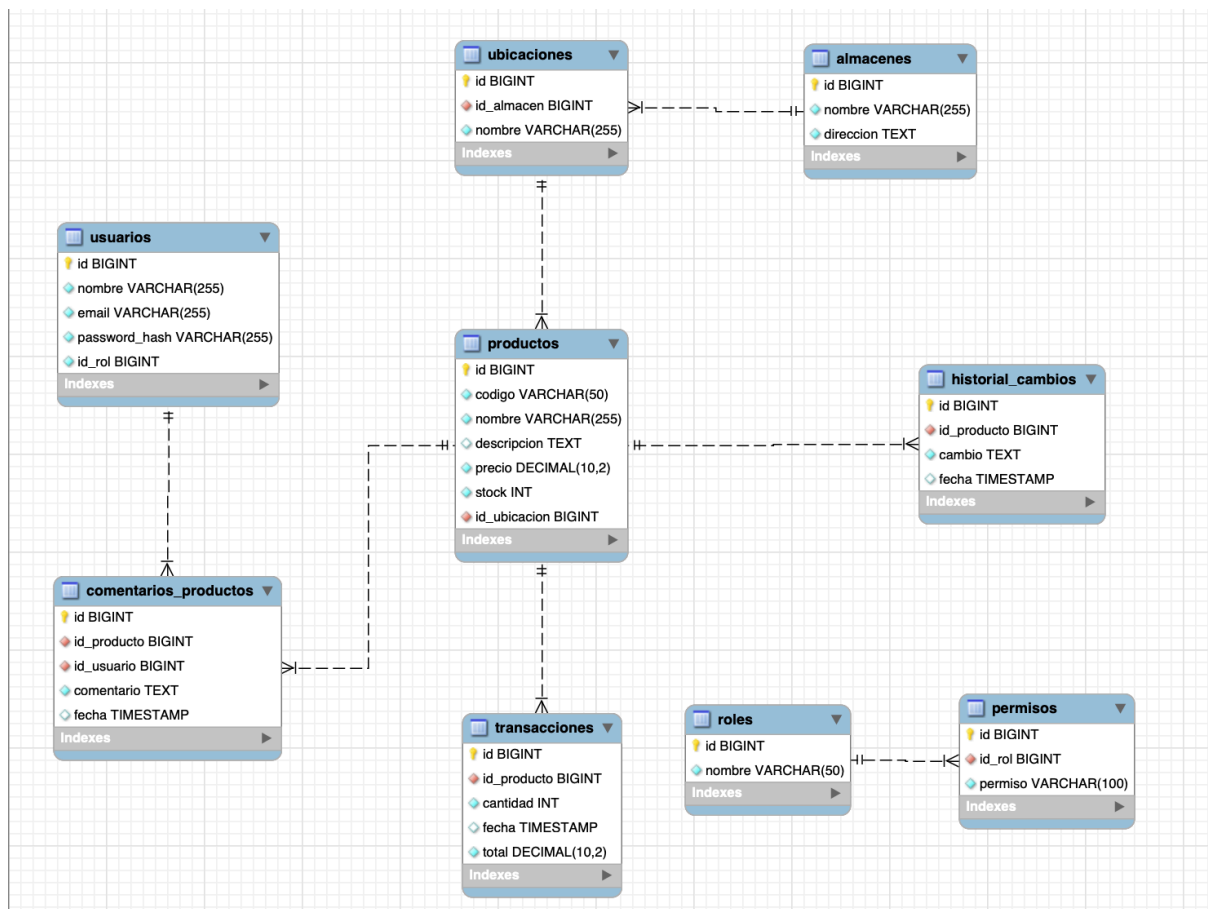
## Bases de Datos III

### Proyecto Final Bases de Datos

Entrega # 1: 17 de Marzo, 2025

#### 1. Toma de Requerimientos

a. Diagramas E-R y diseño lógico del sistema.



#### 2. Diseño de la base de datos

a. Creación de tablas, relaciones, funciones, procedimientos y/o triggers en MySQL.

-- Crear la base de datos

```
CREATE DATABASE IF NOT EXISTS gestion_inventarios;  
USE gestion_inventarios;
```

-- Tabla de almacenes

```
CREATE TABLE almacenes (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    nombre VARCHAR(255) NOT NULL,  
    direccion TEXT NOT NULL  
);
```

-- Tabla de ubicaciones dentro de almacenes

```
CREATE TABLE ubicaciones (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    id_almacen BIGINT NOT NULL,  
    nombre VARCHAR(255) NOT NULL,  
    FOREIGN KEY (id_almacen) REFERENCES almacenes(id) ON DELETE CASCADE  
);
```

-- Tabla de productos

```
CREATE TABLE productos (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    codigo VARCHAR(50) UNIQUE NOT NULL,  
    nombre VARCHAR(255) NOT NULL,  
    descripcion TEXT,  
    precio DECIMAL(10,2) NOT NULL,  
    stock INT NOT NULL,  
    id_ubicacion BIGINT NOT NULL,  
    FOREIGN KEY (id_ubicacion) REFERENCES ubicaciones(id) ON DELETE CASCADE  
);
```

-- Tabla de transacciones (ventas simuladas)

```
CREATE TABLE transacciones (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    id_producto BIGINT NOT NULL,  
    cantidad INT NOT NULL,  
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    total DECIMAL(10,2) NOT NULL,  
    FOREIGN KEY (id_producto) REFERENCES productos(id) ON DELETE CASCADE  
);
```

-- Tabla de usuarios

```
CREATE TABLE usuarios (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    nombre VARCHAR(255) NOT NULL,  
    email VARCHAR(255) UNIQUE NOT NULL,  
    password_hash VARCHAR(255) NOT NULL,  
    id_rol BIGINT NOT NULL  
);
```

-- Tabla de roles

```
CREATE TABLE roles (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    nombre VARCHAR(50) NOT NULL UNIQUE  
);
```

-- Tabla de permisos

```
CREATE TABLE permisos (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    id_rol BIGINT NOT NULL,  
    permiso VARCHAR(100) NOT NULL,  
    FOREIGN KEY (id_rol) REFERENCES roles(id) ON DELETE CASCADE  
);
```

-- Tabla de historial de cambios de productos

```
CREATE TABLE historial_cambios (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    id_producto BIGINT NOT NULL,  
    cambio TEXT NOT NULL,  
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (id_producto) REFERENCES productos(id) ON DELETE CASCADE  
);
```

-- Tabla de comentarios sobre productos

```
CREATE TABLE comentarios_productos (  
    id BIGINT AUTO_INCREMENT PRIMARY KEY,  
    id_producto BIGINT NOT NULL,  
    id_usuario BIGINT NOT NULL,  
    comentario TEXT NOT NULL,  
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (id_producto) REFERENCES productos(id) ON DELETE CASCADE,  
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id) ON DELETE CASCADE  
);
```

-- Insertar roles por defecto

```
INSERT INTO roles (nombre) VALUES ('Administrador'), ('Operador');
```

-- Procedimiento para agregar productos

```
DELIMITER $$  
CREATE PROCEDURE agregar_producto (  
    IN p_codigo VARCHAR(50),  
    IN p_nombre VARCHAR(255),  
    IN p_descripcion TEXT,  
    IN p_precio DECIMAL(10,2),  
    IN p_stock INT,  
    IN p_id_ubicacion BIGINT  
)  
BEGIN  
    INSERT INTO productos (codigo, nombre, descripcion, precio, stock, id_ubicacion)  
    VALUES (p_codigo, p_nombre, p_descripcion, p_precio, p_stock, p_id_ubicacion);  
END$$  
DELIMITER ;
```

-- Procedimiento para actualizar el stock de un producto

```
DELIMITER $$  
CREATE PROCEDURE actualizar_stock (  
    IN p_id_producto BIGINT,  
    IN p_nuevo_stock INT  
)  
;
```

```
BEGIN
  UPDATE productos SET stock = p_nuevo_stock WHERE id = p_id_producto;
END$$
DELIMITER ;

-- Procedimiento para registrar una venta
DELIMITER $$
CREATE PROCEDURE registrar_venta (
  IN p_id_producto BIGINT,
  IN p_cantidad INT
)
BEGIN
  DECLARE v_precio DECIMAL(10,2);
  SELECT precio INTO v_precio FROM productos WHERE id = p_id_producto;
  INSERT INTO transacciones (id_producto, cantidad, total)
  VALUES (p_id_producto, p_cantidad, v_precio * p_cantidad);
  UPDATE productos SET stock = stock - p_cantidad WHERE id = p_id_producto;
END$$
DELIMITER ;

-- Trigger para evitar stock negativo
DELIMITER $$
CREATE TRIGGER before_venta_insert
BEFORE INSERT ON transacciones
FOR EACH ROW
BEGIN
  DECLARE v_stock INT;
  SELECT stock INTO v_stock FROM productos WHERE id = NEW.id_producto;
  IF v_stock < NEW.cantidad THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stock insuficiente';
  END IF;
END$$
DELIMITER ;

-- Trigger para registrar eliminación de productos en historial
DELIMITER $$
CREATE TRIGGER after_producto_delete
AFTER DELETE ON productos
FOR EACH ROW
BEGIN
  INSERT INTO historial_cambios (id_producto, cambio)
  VALUES (OLD.id, 'Producto eliminado');
END$$
DELIMITER ;

-- Función para calcular el valor total del inventario
DELIMITER $$
CREATE FUNCTION calcular_valor_total_inventario()
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
  DECLARE total DECIMAL(10,2);
```

```
SELECT SUM(precio * stock) INTO total FROM productos;  
RETURN total;  
END$$  
DELIMITER ;
```

-- Procedimiento para agregar usuarios

```
DELIMITER $$  
CREATE PROCEDURE agregar_usuario (  
    IN p_nombre VARCHAR(255),  
    IN p_email VARCHAR(255),  
    IN p_password_hash VARCHAR(255),  
    IN p_id_rol BIGINT  
)  
BEGIN  
    INSERT INTO usuarios (nombre, email, password_hash, id_rol)  
    VALUES (p_nombre, p_email, p_password_hash, p_id_rol);  
END$$  
DELIMITER ;
```

-- Procedimiento para actualizar el rol de un usuario

```
DELIMITER $$  
CREATE PROCEDURE actualizar_rol_usuario (  
    IN p_id_usuario BIGINT,  
    IN p_nuevo_id_rol BIGINT  
)  
BEGIN  
    UPDATE usuarios SET id_rol = p_nuevo_id_rol WHERE id = p_id_usuario;  
END$$  
DELIMITER ;
```

-- Procedimiento para eliminar un usuario

```
DELIMITER $$  
CREATE PROCEDURE eliminar_usuario (  
    IN p_id_usuario BIGINT  
)  
BEGIN  
    DELETE FROM usuarios WHERE id = p_id_usuario;  
END$$  
DELIMITER ;
```

-- Procedimiento para listar usuarios y sus roles

```
DELIMITER $$  
CREATE PROCEDURE listar_usuarios_rols()  
BEGIN  
    SELECT u.id, u.nombre, u.email, r.nombre AS rol FROM usuarios u  
    JOIN roles r ON u.id_rol = r.id;  
END$$  
DELIMITER ;
```

-- Procedimiento para asignar un permiso a un rol

```
DELIMITER $$
```

```
CREATE PROCEDURE asignar_permiso (  
    IN p_id_rol BIGINT,  
    IN p_permiso VARCHAR(100)  
)  
BEGIN  
    INSERT INTO permisos (id_rol, permiso) VALUES (p_id_rol, p_permiso);  
END$$  
DELIMITER ;
```

### Reportes Generales

-- Reporte de inventario general

```
DELIMITER $$  
CREATE PROCEDURE reporte_inventario_general()  
BEGIN  
    SELECT p.codigo, p.nombre, p.stock, p.precio, (p.stock * p.precio) AS valor_total  
    FROM productos p;  
END$$  
DELIMITER ;
```

-- Reporte de productos por ubicación

```
DELIMITER $$  
CREATE PROCEDURE reporte_productos_por_ubicacion()  
BEGIN  
    SELECT a.nombre AS almacen, u.nombre AS ubicacion, p.nombre AS producto, p.stock  
    FROM productos p  
    JOIN ubicaciones u ON p.id_ubicacion = u.id  
    JOIN almacenes a ON u.id_almacen = a.id  
    ORDER BY a.nombre, u.nombre;  
END$$  
DELIMITER ;
```

-- Reporte de ventas simuladas

```
DELIMITER $$  
CREATE PROCEDURE reporte_ventas_simuladas()  
BEGIN  
    SELECT t.id, p.nombre AS producto, t.cantidad, t.fecha, t.total  
    FROM transacciones t  
    JOIN productos p ON t.id_producto = p.id  
    ORDER BY t.fecha DESC;  
END$$  
DELIMITER ;
```

-- Reporte de productos con bajo stock

```
DELIMITER $$  
CREATE PROCEDURE reporte_productos_bajo_stock(IN nivel_minimo INT)  
BEGIN  
    SELECT p.codigo, p.nombre, p.stock  
    FROM productos p  
    WHERE p.stock < nivel_minimo  
    ORDER BY p.stock ASC;  
END$$
```

DELIMITER ;

-- Reporte de usuarios y sus roles

DELIMITER \$\$

CREATE PROCEDURE reporte\_usuarios\_rols()

BEGIN

SELECT u.id, u.nombre, u.email, r.nombre AS rol

FROM usuarios u

JOIN roles r ON u.id\_rol = r.id;

END\$\$

DELIMITER ;

## b. Configuración de bases de datos en MongoDB.

Se crea en MongoDB Atlas la base de datos llamada **ProyectoGestionInventariosBD3**. En ella se pretende llevar un registro histórico de los cambios en los productos, de los comentarios realizados por los operadores, y de las transacciones pasadas. Para ello se agregan estas 3 colecciones.

DINA'S ORG - 2025-03-02 > PROJECT 0 > DATABASES

ClusterO

VERSION: 8.0.5 REGION: AWS N. Virginia (us-east-1)

Overview Real Time Metrics Collections Atlas Search Performance Advisor Online Archive Cmd Line Tools Infrastructure As C

DATABASES: 4 COLLECTIONS: 12

+ Create Database

Q Search Namespaces

ProjectoGestionInventari...

cambiosenproductos

comentariosoperadores

transaccionespasadas

ProjectoGestionInventariosBD3

LOGICAL DATA SIZE: 0B STORAGE SIZE: 12KB INDEX SIZE: 12KB TOTAL COLLECTIONS: 3

CREATE COLLECTION

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
cambiosenproductos	0	0B	0B	4KB	1	4KB	4KB
comentariosoperadores	0	0B	0B	4KB	1	4KB	4KB
transaccionespasadas	0	0B	0B	4KB	1	4KB	4KB

## Explicacion del diseño y configuracion:

### 1. Diseño de la Base de Datos

La base de datos gestion\_inventarios está diseñada para administrar almacenes, productos, transacciones, usuarios y roles. Se estructura en distintas tablas con relaciones bien definidas.

### Tablas y Relaciones

#### Almacenes y Ubicaciones

**Almacenes:** Contiene información sobre los almacenes.

ubicaciones: Define ubicaciones dentro de cada almacén y tiene una relación con almacenes.

Productos

**Productos:** Contiene información sobre cada producto, incluyendo precio, stock y ubicación.

Relación: Cada producto pertenece a una ubicación (id\_ubicacion).

Transacciones (Ventas Simuladas)

**Transacciones:** Registra cada venta, asociada a un producto (id\_producto).

Usuarios, Roles y Permisos

**Usuarios:** Almacena los datos de los usuarios, incluyendo el rol asignado.

roles: Define los distintos roles en el sistema (Administrador, Operador, etc.).

permisos: Relaciona roles con permisos específicos.

Historial y Comentarios

**historial\_cambios:** Registra modificaciones o eliminaciones de productos.

comentarios\_productos: Permite a los usuarios dejar comentarios sobre los productos.

## 2. Creación de Tablas

Cada tabla tiene claves primarias (PRIMARY KEY), y en algunas se establecen relaciones mediante claves foráneas (FOREIGN KEY) con la opción ON DELETE CASCADE para eliminar datos relacionados automáticamente.

Ejemplo:

I

Copy

Edit

```
CREATE TABLE productos (  
  id BIGINT AUTO_INCREMENT PRIMARY KEY,  
  codigo VARCHAR(50) UNIQUE NOT NULL,  
  nombre VARCHAR(255) NOT NULL,  
  descripcion TEXT,  
  precio DECIMAL(10,2) NOT NULL,  
  stock INT NOT NULL,  
  id_ubicacion BIGINT NOT NULL,  
  FOREIGN KEY (id_ubicacion) REFERENCES ubicaciones(id) ON DELETE CASCADE  
);
```

Aquí:

id es la clave primaria.

codigo es único para cada producto.

id\_ubicacion es clave foránea vinculada a ubicaciones(id).

## 3. Procedimientos Almacenados

Los procedimientos almacenados facilitan la gestión de productos, usuarios y transacciones.

Ejemplo 1: Agregar un producto

sql

Copy

Edit



```
DELIMITER $$
CREATE PROCEDURE agregar_producto (
    IN p_codigo VARCHAR(50),
    IN p_nombre VARCHAR(255),
    IN p_descripcion TEXT,
    IN p_precio DECIMAL(10,2),
    IN p_stock INT,
    IN p_id_ubicacion BIGINT
)
BEGIN
    INSERT INTO productos (codigo, nombre, descripcion, precio, stock, id_ubicacion)
    VALUES (p_codigo, p_nombre, p_descripcion, p_precio, p_stock, p_id_ubicacion);
END$$
DELIMITER ;
```

Objetivo: Inserta un nuevo producto en la base de datos.  
Parámetros: Código, nombre, descripción, precio, stock y ubicación.

Ejemplo 2: Registrar una venta

sql

Copy

Edit

```
DELIMITER $$
```

```
CREATE PROCEDURE registrar_venta (
    IN p_id_producto BIGINT,
    IN p_cantidad INT
)
BEGIN
    DECLARE v_precio DECIMAL(10,2);
    SELECT precio INTO v_precio FROM productos WHERE id = p_id_producto;
    INSERT INTO transacciones (id_producto, cantidad, total)
    VALUES (p_id_producto, p_cantidad, v_precio * p_cantidad);
    UPDATE productos SET stock = stock - p_cantidad WHERE id = p_id_producto;
END$$
DELIMITER ;
```

Objetivo: Registra una venta y actualiza el stock.

Lógica:

Obtiene el precio del producto.

Inserta la venta en transacciones.

Reduce el stock en productos.

#### 4. Triggers

Los triggers ejecutan acciones automáticamente ante ciertos eventos.

Ejemplo 1: Evitar stock negativo antes de una venta

sql

Copy

Edit

```
DELIMITER $$
CREATE TRIGGER before_venta_insert
BEFORE INSERT ON transacciones
FOR EACH ROW
BEGIN
    DECLARE v_stock INT;
    SELECT stock INTO v_stock FROM productos WHERE id = NEW.id_producto;
    IF v_stock < NEW.cantidad THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stock insuficiente';
    END IF;
END$$
DELIMITER ;
```

Antes de insertar una venta, verifica que haya suficiente stock.  
Si no hay stock suficiente, lanza un error.

Ejemplo 2: Registrar eliminación de productos en el historial

```
sql
Copy
Edit
DELIMITER $$
CREATE TRIGGER after_producto_delete
AFTER DELETE ON productos
FOR EACH ROW
BEGIN
    INSERT INTO historial_cambios (id_producto, cambio)
    VALUES (OLD.id, 'Producto eliminado');
END$$
DELIMITER ;
```

Después de eliminar un producto, guarda un registro en historial\_cambios.

## 5. Función para Calcular Valor Total del Inventario

Las funciones devuelven valores específicos, útiles para reportes.

```
Copy
Edit
DELIMITER $$
CREATE FUNCTION calcular_valor_total_inventario()
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE total DECIMAL(10,2);
    SELECT SUM(precio * stock) INTO total FROM productos;
    RETURN total;
END$$
DELIMITER ;
```

Retorna la suma total del inventario (precio \* stock).

## 6. Reportes con Procedimientos

Se han creado reportes para consultas comunes.

### Ejemplo 1: Reporte de Inventario General

sql

Copy

Edit

DELIMITER \$\$

CREATE PROCEDURE reporte\_inventario\_general()

BEGIN

SELECT p.codigo, p.nombre, p.stock, p.precio, (p.stock \* p.precio) AS valor\_total  
FROM productos p;

END\$\$

DELIMITER ;

Objetivo: Muestra código, nombre, stock, precio y valor total de cada producto.

### Ejemplo 2: Reporte de Productos con Bajo Stock

sql

Copy

Edit

DELIMITER \$\$

CREATE PROCEDURE reporte\_productos\_bajo\_stock(IN nivel\_minimo INT)

BEGIN

SELECT p.codigo, p.nombre, p.stock  
FROM productos p  
WHERE p.stock < nivel\_minimo  
ORDER BY p.stock ASC;

END\$\$

DELIMITER ;

Objetivo: Lista productos con stock menor al valor mínimo ingresado.

## FASE 2: Funcionalidades avanzadas

### 1. Optimización y replicación

#### a) Implementación de partición horizontal

Se crearon dos tablas separadas que simulan una partición horizontal según el almacén:

CREATE TABLE productos\_almacen1 LIKE productos;

CREATE TABLE productos\_almacen2 LIKE productos;

Estas pueden almacenar productos separados por tipo o ubicación lógica.

## b) Implementación de partición vertical

Se creó una tabla adicional productos\_detalle que complementa la información de productos, dividiendo los atributos entre dos tablas:

```
CREATE TABLE productos_detalle (  
    id BIGINT PRIMARY KEY,  
    descripcion TEXT,  
    id_ubicacion BIGINT NOT NULL,  
    FOREIGN KEY (id) REFERENCES productos(id) ON DELETE CASCADE,  
    FOREIGN KEY (id_ubicacion) REFERENCES ubicaciones(id) ON DELETE CASCADE  
);
```

## c) Configuración de replicación en MySQL (Docker)

### Pasos realizados:

1. Se usó docker-compose.yml con dos servicios: mysql-master y mysql-slave.
2. Se configuró el master para que use binlogs y permita replicación:  
command: --server-id=1 --log-bin=mysql-bin --binlog-do-db=gestion\_inventarios  
--binlog\_format=ROW
3. En el slave:  
command: --server-id=2 --relay-log=relay-log --read-only=1 --binlog\_format=ROW

En **mysql-master** se creó el usuario replica:

```
CREATE USER 'replica'@'%' IDENTIFIED WITH mysql_native_password BY 'replica123';
```

```
GRANT REPLICATION SLAVE ON *.* TO 'replica'@'%';
```

```
FLUSH PRIVILEGES;
```

4. SHOW MASTER STATUS;

En **mysql-slave** se configuró:

```
CHANGE MASTER TO
```

```
MASTER_HOST='mysql-master',
```

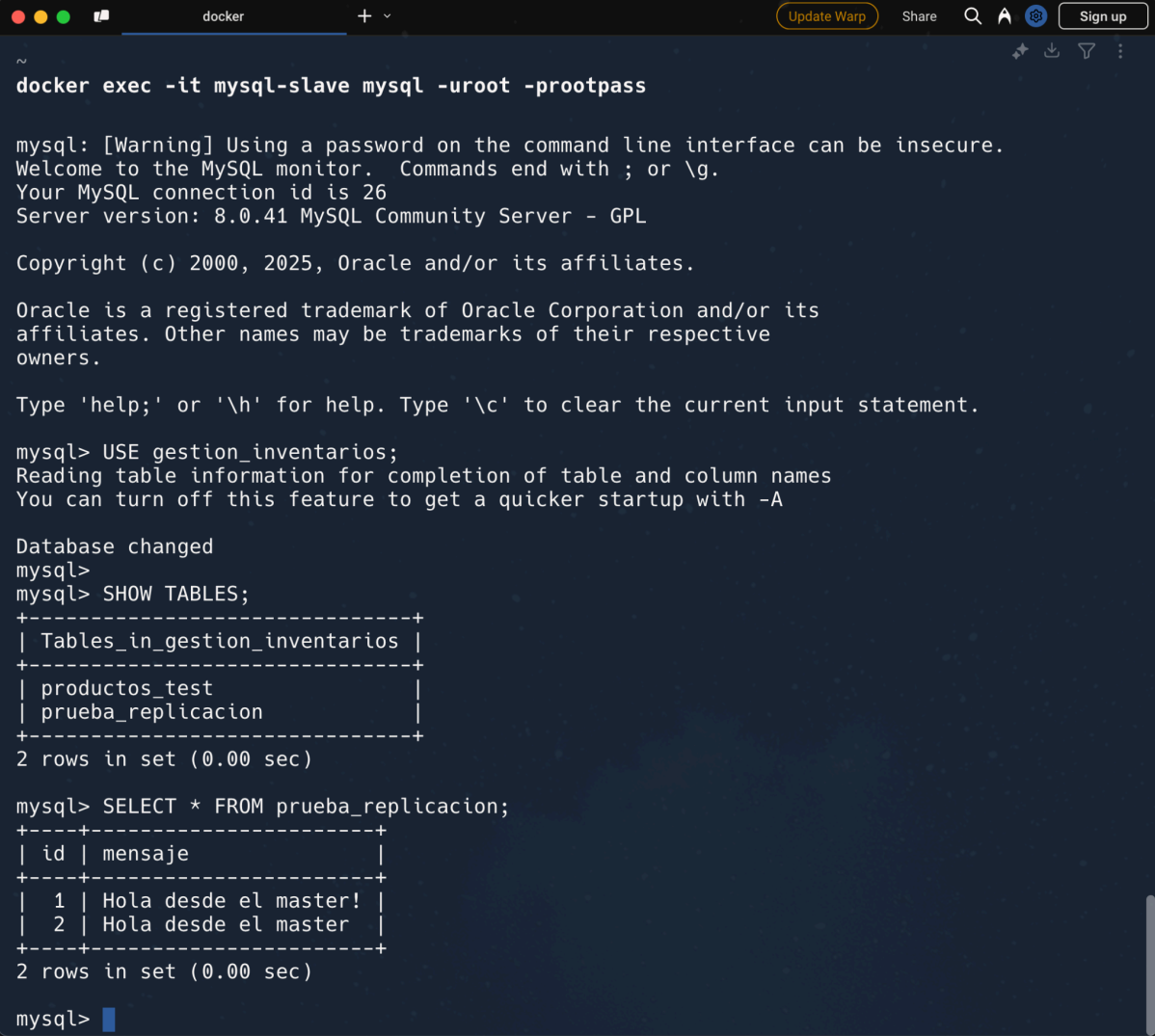
```
MASTER_USER='replica',
```

```
MASTER_PASSWORD='replica123',
```

```
MASTER_LOG_FILE='mysql-bin.000004',
```

```
MASTER_LOG_POS=547;
```

5. START SLAVE;
6. Se verificó con:  
SHOW SLAVE STATUS\G  
Y ambos **Slave\_IO\_Running** y **Slave\_SQL\_Running** en **Yes**
7. Se realizó prueba de replicación insertando desde el master:  
INSERT INTO prueba\_replicacion (mensaje) VALUES ('Hola desde el master');  
Y visualizándolo en el slave con:  
SELECT \* FROM prueba\_replicacion;



```
docker exec -it mysql-slave mysql -uroot -prootpass

mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 8.0.41 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE gestion_inventarios;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
mysql> SHOW TABLES;
+-----+
| Tables_in_gestion_inventarios |
+-----+
| productos_test                  |
| prueba_replicacion              |
+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM prueba_replicacion;
+-----+-----+
| id | mensaje                |
+-----+-----+
| 1  | Hola desde el master!  |
| 2  | Hola desde el master   |
+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

## 2. Consultas avanzadas y reportes

### a) Reportes con filtros avanzados (procedimientos):

CALL reporte\_inventario\_general();

CALL reporte\_productos\_por\_ubicacion();

```
CALL reporte_ventas_simuladas();  
CALL reporte_productos_bajo_stock(10);  
CALL reporte_usuarios_rolles();
```

#### **b) Vistas agrupadas:**

```
CREATE VIEW vista_productos_por_ubicacion AS  
SELECT a.nombre AS nombre_almacen, u.nombre AS nombre_ubicacion, p.nombre AS  
nombre_producto, p.stock  
FROM productos p  
JOIN ubicaciones u ON p.id_ubicacion = u.id  
JOIN almacenes a ON u.id_almacen = a.id;
```

```
CREATE VIEW vista_resumen_ventas AS  
SELECT p.nombre AS producto, COUNT(t.id) AS total_transacciones, SUM(t.cantidad) AS  
total_vendido, SUM(t.total) AS ingresos_totales  
FROM transacciones t  
JOIN productos p ON t.id_producto = p.id  
GROUP BY p.nombre;
```

### **3. Gestión de usuarios y seguridad**

#### **a) Roles y permisos implementados**

Tablas creadas:

- usuarios
- roles
- permisos

Procedimientos usados:

```
CALL agregar_usuario(...);  
CALL actualizar_rol_usuario(...);  
CALL eliminar_usuario(...);  
CALL asignar_permiso(...);  
CALL listar_usuarios_roles();
```

### **4. Almacenamiento NoSQL**

Proyecto adjunto en github

## 5. Presentación final y documentación completa

### Arquitectura General del Sistema

El sistema utiliza dos tecnologías clave para gestionar datos: MySQL y MongoDB. MySQL es un motor de base de datos relacional que se encarga de la gestión principal de los datos y utiliza características como la replicación, que asegura que haya una copia de seguridad sincronizada, y la partición, que nos ayuda a mejorar el rendimiento de las consultas y el almacenamiento de grandes volúmenes de datos.

Por otro lado, MongoDB es una base de datos NoSQL, diseñada para manejar documentos de manera flexible y la usamos para almacenar información como el historial de transacciones, comentarios y datos históricos, lo que nos da una mayor flexibilidad en su gestión.

### Funcionalidades Implementadas

Implementamos varios procedimientos almacenados nos permite realizar acciones clave, como la alta, baja y modificación de productos y usuarios. Además, controla el stock validando las cantidades disponibles antes de registrar cualquier venta. También genera reportes útiles, como los de inventario, ventas y productos con bajo stock.

### 5.3. Base No Relacional (MongoDB)

La base de datos "ProyectoGestionInventariosBD3" en MongoDB Atlas está configurada para almacenar datos no relacionales, específicamente diseñados para llevar un registro histórico de las transacciones, cambios en productos y comentarios de operadores. Esta configuración permite registrar las transacciones pasadas de los productos mediante una API que simula su registro, lo que permite rastrear todas las operaciones realizadas a lo largo del tiempo. Además, se implementa un historial de modificaciones para los productos, permitiendo un seguimiento detallado de los cambios en su información (como precios, descripciones o cantidades).

Por último y para completar la base de datos de MongoDB, se crea una colección para los comentarios de los operadores, donde se registran observaciones y notas relevantes relacionadas con los productos, asegurando un contexto más claro sobre las decisiones o acciones tomadas. Esta estructura facilita una gestión integral y transparente de los datos históricos dentro del sistema de inventarios.

#### **Detalle de MongoDB Atlas:**

- - Base de datos: `ProyectoGestionInventariosBD3`
- - Colecciones definidas
  - `historialCambios`: respaldo o bitácora más detallada o enriquecida.
  - `comentariosOperadores`: seguimiento informal, opiniones, sugerencias.
  - `transaccionesPasadas`: archivar ventas con datos adicionales (geolocalización, dispositivo, etc.).

#### Beneficios:

- Información flexible y variable.
- Almacenar documentos históricos o extendidos sin afectar la estructura rígida de MySQL.

#### Reportes que incluidos:

El sistema genera varios reportes que son útiles para el análisis y la gestión del inventario. Estos incluyen el inventario general con el cálculo de su valor total, la distribución de productos por ubicación o almacén, el reporte de ventas ordenado cronológicamente, alertas por bajo stock y un reporte de usuarios con sus roles respectivos.

#### Conclusión final:

Este sistema de gestión de inventario está diseñado para ser escalable y eficiente. Combina MySQL para la gestión estructurada de MongoDB para manejar información más dinámica y menos estructurada. también, es capaz de soportar las operaciones diarias, generar reportes, realizar auditorías y administrar usuarios con un control adecuado sobre los permisos.