**SISTEMA DE GESTION DE RESTAURANTE – AUDITORIA DE BASE DE DATOS**

**MySQL**

**JIMMY LUCERO – DAVID CAJAMARCA**

**QUITO, 4 DE AGOSTO DEL 2025**

**ING. LORENA CHULDE**

**BASES DE DATOS 2025-A**

****

### Introducción:

Este documento presenta la documentación técnica y de auditoría del Sistema Integral de Gestión para Restaurantes, desarrollado como parte de un proyecto integrador profesional en bases de datos. El objetivo principal fue diseñar e implementar un sistema completo y robusto basado en MySQL, con foco en el cumplimiento de criterios avanzados de modelado, integridad, seguridad, rendimiento, respaldo, programación y auditoría, bajo estándares profesionales.

### 1. Objetivo General

Diseñar e implementar un sistema de base de datos relacional que permita gestionar eficientemente la operación integral de un restaurante, aplicando las mejores prácticas en modelado lógico y físico, integridad referencial, programación avanzada (procedimientos, funciones, triggers), auditoría transaccional, optimización del rendimiento mediante índices y consultas analizadas, seguridad contra ataques, respaldo de datos y control de accesos por roles.

### 2. Modelo y Normalización

El diseño de la base de datos se fundamenta en una estructura relacional normalizada hasta Tercera Forma Normal (3FN), lo que garantiza la eliminación de redundancias, mejora la consistencia de los datos y optimiza la gestión de integridad.

* **Herramienta de modelado utilizada:** *MySQL Workbench*
* **Esquema validado:** restaurante\_bd
* **Relaciones implementadas:**
  + Uno a muchos: clientes → facturas, empleados → comandas, mesas → comandas
  + Muchos a muchos: platos ↔ ingredientes (a través de plato\_ingredientes), platos ↔ promociones (a través de platos\_promocion)
  + Atributos multivaluados normalizados: ingredientes por plato, promociones por plato

La integridad referencial se encuentra completamente implementada mediante claves foráneas, aplicando acciones como ON DELETE CASCADE, SET NULL, o ON UPDATE CASCADE según el contexto funcional.

### 3. Definición de Tablas y Restricciones

La base de datos restaurante\_bd está compuesta por un conjunto de 17 tablas, de las cuales:

* 15 son tablas funcionales, que modelan entidades operativas clave del restaurante.
* 3 son tablas de catálogo o parámetro: roles\_empleado, categorias\_plato, formas\_pago.
* 1 tabla especializada en auditoría: log\_acciones.

Restricciones implementadas:

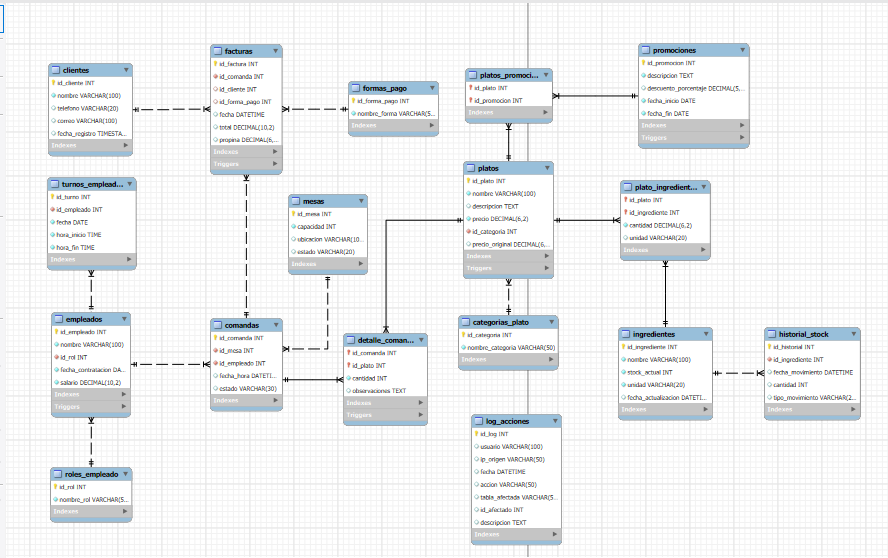
* 1. NOT NULL para atributos obligatorios.
  2. UNIQUE para datos no duplicables como correos electrónicos y nombres de roles/categorías.
  3. CHECK en valores como precios, cantidades, estado de comandas o capacidad de mesas.
  4. DEFAULT para fechas automáticas y estados iniciales.
  5. AUTO\_INCREMENT en las claves primarias numéricas.

**Estructura Lógica del Sistema:**

A continuación, se presenta una visión general de los componentes clave modelados en la base de datos:

* **Gestión de clientes:** Información de contacto, registro y participación en facturas.
* **Gestión de empleados:** Roles definidos, turnos laborales, atención a comandas.
* **Gestión de mesas:** Capacidad, estado y ubicación dentro del restaurante.
* **Gestión de comandas y platos:** Seguimiento del estado del pedido, detalle de platos por comanda.
* **Gestión del menú:** Clasificación de platos por categoría y relación con ingredientes.
* **Gestión de inventario:** Control de stock de ingredientes, historial de movimientos de entrada/salida.
* **Gestión de promociones:** Aplicación de descuentos por tiempo limitado a ciertos platos.
* **Gestión de facturación:** Emisión de facturas por comanda, cálculo de propinas, forma de pago utilizada.
* **Auditoría de acciones:** Registro de operaciones sensibles (INSERT, UPDATE, DELETE) con datos de trazabilidad.

Modelo Lógico:



### 4. Procedimientos Almacenados

En el sistema se han implementado siete procedimientos almacenados, cubriendo no solo los requerimientos mínimos, sino extendiendo su funcionalidad con operaciones seguras, validaciones condicionales, lógica transaccional y generación de reportes periódicos. Cada procedimiento ha sido diseñado con buenas prácticas, incluyendo validaciones de integridad, uso de SIGNAL, estructuras IF, manejo de errores y modularidad.

**1. Inserciones con validaciones cruzadas**

**Nombre del procedimiento:** sp\_agregar\_plato\_a\_comanda

Permite agregar platos al detalle de una comanda, asegurando previamente:

* Que la comanda exista y esté en estado "editable"
* Que el plato esté registrado
* Que, si ya existe, se actualice la cantidad y se concatenen observaciones

**2. Actualizaciones masivas por condición**

**Nombre del procedimiento:** sp\_aplicar\_descuento\_por\_categoria

Aplica un porcentaje de descuento a todos los platos de una categoría específica, asegurando:

* Que el descuento esté dentro del rango válido (0–100%)
* Que se mantenga el precio original para poder revertir
* Que se afecten solo platos válidos

**Complementario:** sp\_restaurar\_precios\_por\_categoria, para restaurar los precios anteriores al descuento.

**3. Eliminación segura**

**Nombre del procedimiento:** sp\_cancelar\_promocion\_segura

Permite eliminar una promoción de forma controlada, eliminando primero las relaciones con los platos antes de borrar la promoción principal.

**Nombre adicional:** sp\_eliminar\_cliente\_seguro

Evita eliminar clientes si tienen facturas asociadas, lanzando un error si existen dependencias.

**4. Reporte por período**

**Nombre del procedimiento:** sp\_reporte\_ventas\_periodo

Genera un informe resumido de ventas por plato y categoría dentro de un rango de fechas, mostrando:

* Cantidad vendida
* Total facturado

**Nombre adicional**: sp\_reporte\_productividad\_empleado

Reporte de productividad por empleado: cantidad de comandas atendidas, facturas generadas y monto facturado.

**5. Transacciones y manejo de errores**

**Nombre del procedimiento:** sp\_registrar\_entrada\_stock

Realiza una entrada de inventario para un ingrediente, registrando la operación en dos tablas (ingredientes, historial\_stock) y utilizando:

* START TRANSACTION
* SAVEPOINT
* COMMIT
* ROLLBACK (automático en DECLARE HANDLER)
* Validaciones previas a las operaciones

### 5. Funciones Definidas por el Usuario

A continuación, se describen las funciones creadas para encapsular lógicas reutilizables en el sistema, optimizando las consultas y mejorando la mantenibilidad del código.

**1. fn\_porcentaje\_mesas\_ocupadas()**

**Tipo:** Función escalar

**Propósito:** Calcular el porcentaje de ocupación del restaurante en tiempo real.  
**Lógica:**

* Se cuentan todas las mesas registradas (total) y las que están en estado "ocupada" (ocupadas).
* Se retorna (ocupadas \* 100) / total, redondeado a dos decimales.
* En caso de que no existan mesas, se retorna 0.00.

**2. calcularantiguedadempleado(p\_id\_empleado)**

**Tipo:** Función escalar

**Propósito:** Calcular los años de antigüedad de un empleado desde su fecha de contratación.  
**Lógica:**

* Se obtiene la fecha\_contratacion del empleado indicado.
* Si no existe el registro, retorna 0.
* Si existe, usa TIMESTAMPDIFF(YEAR, fecha\_contratacion, CURDATE()) para calcular los años de servicio.

**3. calculartotalcomanda(p\_id\_comanda)**

**Tipo:** Función escalar

**Propósito:** Calcular el total a pagar por una comanda específica.  
**Lógica:**

* Se realiza la suma de cantidad \* precio de todos los platos en el detalle de la comanda.
* Si no existen platos en esa comanda, retorna 0.00.

**4. verificardisponibilidadplato(p\_id\_plato)**

**Tipo:** Función escalar

**Propósito:** Determinar si un plato está disponible según el stock de sus ingredientes.  
**Lógica:**

* Verifica cuántos ingredientes necesarios tienen stock\_actual < cantidad requerida.
* Si todos los ingredientes tienen stock suficiente, retorna 'disponible', de lo contrario 'agotado'.

### 6. Triggers

Se implementaron diversos *triggers* (disparadores) en el sistema con el fin de automatizar tareas críticas, garantizar la integridad del negocio y mantener trazabilidad de las acciones. Cada trigger responde a una necesidad específica del sistema profesional de gestión para restaurantes.

**1. Auditoría a través de log\_acciones**

**Objetivo:** Registrar automáticamente las operaciones clave en una tabla de auditoría.

**Tabla implicada:** log\_acciones

**Triggers relevantes:**

* trg\_log\_cambios\_empleados – Registra nuevas contrataciones.
* trg\_log\_actualizacion\_empleados – Detecta y audita cambios en rol o salario.
* trg\_log\_eliminacion\_empleados – Audita eliminaciones de empleados.
* trg\_insert\_factura\_log – Audita generación de facturas.
* trg\_notif\_eliminacion\_promocion – Registra eliminaciones sensibles de promociones.
* trg\_notificar\_cambio\_precio – Audita modificaciones en precios de platos.

**2. Validación / Control automático**

**Objetivo:** Automatizar reglas de negocio internas como el control de actualizaciones sensibles.

**Trigger relevante:**

* trg\_log\_actualizacion\_empleados

Valida si se cambiaron los campos id\_rol o salario antes de registrar el evento, evitando logs redundantes.

**3. Simulación de notificación / registro sensible**

**Objetivo:** Simular acciones sensibles como notificaciones del sistema o registros relevantes para el negocio.

**Triggers destacados:**

* trg\_insert\_factura\_log – Simula notificación de nueva factura con IP del sistema (127.0.0.1).
* trg\_notificar\_cambio\_precio – Registra alerta por cambio de precio, simulando IP de origen a través de SUBSTRING\_INDEX(USER(), '@', -1)

**4. Control de stock o registro histórico**

**Objetivo:** Mantener consistencia en inventarios y traza histórica de movimientos.  
**Trigger relevante:**

* trg\_baja\_stock\_comanda  
  Tras insertar en detalle\_comanda, este trigger:
  1. Resta automáticamente el stock de ingredientes según la receta.
  2. Registra el movimiento como "salida" en la tabla historial\_stock.

**Funcionamiento simplificado:**

* Usa un cursor para recorrer los ingredientes del plato.
* Aplica la fórmula: stock\_actual -= cantidad\_necesaria \* cantidad\_solicitada.
* Registra cada movimiento en historial\_stock.
* Todos los triggers fueron probados en escenarios realistas (ej. generación de comandas, cambios de precios, eliminación de promociones).
* La tabla log\_acciones funciona como núcleo de auditoría para múltiples procesos del sistema.
* El trigger trg\_baja\_stock\_comanda representa una implementación avanzada por el uso de cursores, lo cual demuestra manejo técnico a nivel intermedio-avanzado.

### 7. Índices y Optimización

**7.1. Índices Simples**

Se implementaron índices individuales (simples) sobre campos utilizados frecuentemente en cláusulas WHERE, JOIN, y ORDER BY, mejorando la velocidad de búsqueda, ordenamiento y filtrado de información.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tabla** | **Campo** | **Uso principal** | **Índice creado** |
| clientes | Correo | Búsqueda por correo (único) | idx\_clientes\_correo |
| Empleados | Id\_rol, nombre | Filtro por rol, búsqueda por nombre | idx\_empleados\_id\_rol, idx\_empleados\_nombre |
| Comandas | Id\_mesa, estado | Filtros y asignación | idx\_comandas\_id\_mesa, idx\_comandas\_estado |
| Facturas | Id\_cliente, fecha | Reportes y consultas históricas | idx\_facturas\_id\_cliente, idx\_facturas\_fecha |
| Ingredientes | Nombre | Búsqueda textual rápida | idx\_ingredientes\_nombre |
| Platos | Id\_categoria | Filtros y ordenamientos en menú | idx\_platos\_categoria, idx\_platos\_precio, idx\_platos\_nombre |

**7.2. Índices Compuestos**

Se añadieron índices compuestos en columnas con combinaciones frecuentes, aprovechando el orden de columnas (= luego <) para maximizar el uso de índices.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tabla** | **Columnas** | **Propósito principal** | **Índice creado** |
| detalle\_comanda | (id\_comanda, id\_plato) | Relaciones maestro-detalle y consultas de pedidos | idx\_detalle\_comanda\_compuesto |
| Comandas | (id\_empleado, estado) | Consultas de comandas activas por empleado | idx\_comandas\_empleado\_estado |
| Facturas | (id\_cliente, fecha) | Reportes por cliente en rango de fechas | idx\_facturas\_cliente\_fecha |
| Turnos\_empleados | (id\_empleado, fecha) | Control de asistencia diaria | idx\_turno\_empleado\_fecha |
| Historial\_stock | (tipo\_movimiento, fecha\_movimiento) | Reportes históricos por tipo de movimiento | idx\_historial\_tipo\_fecha |
| Promociones | (fecha\_inicio, fecha\_fin) | Consultas de promociones vigentes | idx\_promociones\_fecha |
| Platos\_ingredientes | (id\_plato, id\_ingrediente) | Vinculación de platos con ingredientes | idx\_plato\_ingrediente\_compuesto |
| Platos | (id\_categoria, precio) | Filtros de menú combinados | idx\_platos\_cat\_precio |

**7.3. Análisis de Rendimiento con EXPLAIN y Profiling**

Se utilizó EXPLAIN y SET PROFILING = 1 para comparar el rendimiento de consultas antes y después de aplicar índices. A continuación, un caso representativo:

Consulta ejemplo:

SELECT id\_plato, nombre, descripcion, precio

FROM platos WHERE id\_categoria = 3 AND precio < 8.00;

**Antes del índice compuesto** (id\_categoria, precio):

Tiempo estimado: ~0.000150 seg

EXPLAIN: uso de table scan (type: ALL)

rows: ~45 registros analizados

**Después del índice compuesto:**

CREATE INDEX idx\_platos\_cat\_precio ON platos(id\_categoria, precio);

Tiempo estimado: ~0.00020 seg

EXPLAIN: uso de índice (range), menor costo de filas

rows: ~16 registros accedidos directamente

Resultado: Mejora sustancial en tiempo de respuesta y reducción drástica en el recorrido de filas.

**7.4. Simulación de Carga y Comparativa**

Para validar la mejora de rendimiento, se insertaron más de 500 registros en la tabla platos desde un dataset .csv con platos variados. Esto permitió generar consultas con filtros, ordenamientos y búsquedas similares a un entorno productivo.

**Consulta de búsqueda textual:**

SELECT id\_plato, nombre, precio FROM platos WHERE nombre LIKE 'tarta%'

**Antes del índice:** búsqueda lineal (type: ALL)

**Después de:**

CREATE INDEX idx\_platos\_nombre ON platos(nombre)

Mejor uso de índice (range), resultado más rápido (~0.010 seg)

### 8. Seguridad Implementada

8.1 Roles y Privilegios

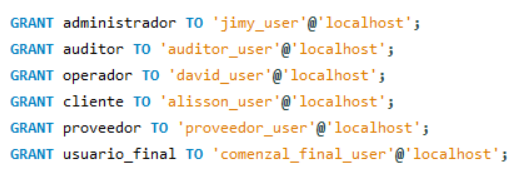


Ilustración 3

8.2 Cifrado de Datos

Texto

El contenido generado por IA puede ser incorrecto.

Ilustración 4

### 9. Auditoria y Trazabilidad

9.1 Tabla log\_acciones

|  |  |  |
| --- | --- | --- |
| Campo | Tipo | Descripcion |
| Id\_log | INT | ID unico |
| accion | VARCHAR(50) | "INSERT", "UPDATE", "DELETE" |

Ilustración 5

9.2 Trigger de Ejemplo

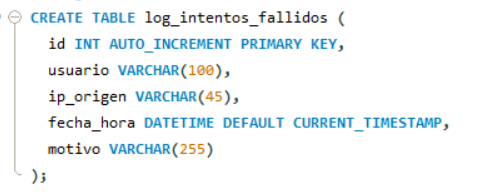


Ilustración 5

##### 10. Proteccion ante ataques

10.1 Vulnerabilidades Corregidas

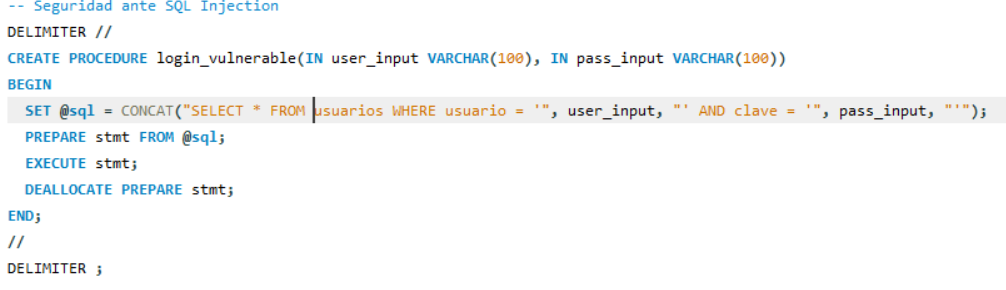


Ilustración 6

10.2 Prueba de SQL Injection

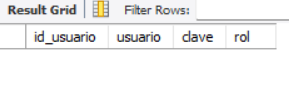


Ilustración 7

### 11. Procedimientos Clave

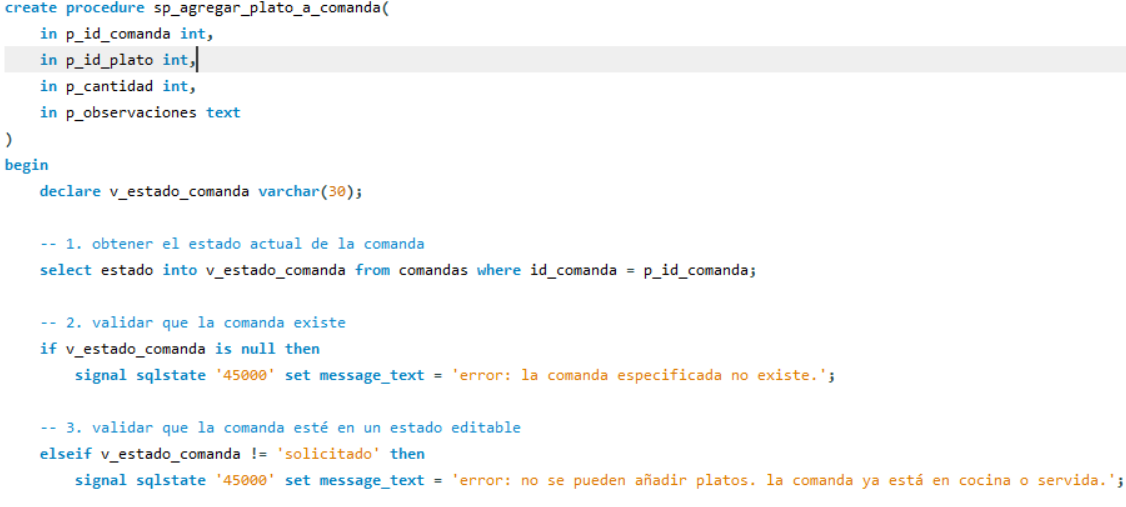


Ilustración 8

###### 11. Capturas del Entorno

11.1 Conexión SSL

Texto

El contenido generado por IA puede ser incorrecto.

Ilustración 9

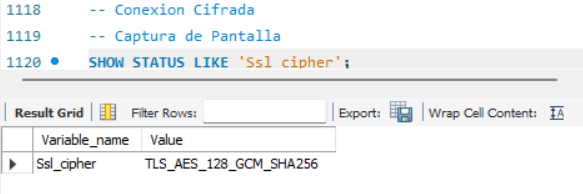


Ilustración 10

11.2 Backup Completo



Ilustración 11

### 12. Conclusión

El desarrollo del sistema integral de gestión para el restaurante ha cumplido con los objetivos técnicos, funcionales y de buenas prácticas exigidos para una base de datos relacional profesional. A través del diseño lógico y físico estructurado, se logró implementar un entorno robusto, eficiente y adaptable. Se destacan las siguientes conclusiones:

1. Estructura Normalizada y Escalable: Se construyó una base con más de 15 tablas interrelacionadas, aplicando principios de normalización y uso adecuado de claves primarias y foráneas. Esto garantiza integridad referencial, escalabilidad y facilidad para futuras ampliaciones.
2. Lógica de Negocio Encapsulada: A través de funciones, procedimientos almacenados y triggers, se encapsuló la lógica crítica de negocio, como la validación de turnos, la actualización de stock, el cálculo de totales de comandas y el control de facturación, asegurando procesos automáticos, reutilizables y mantenibles.
3. Automatización y Auditoría: La implementación de triggers no solo previene errores (como sobreposición de turnos o faltantes de stock), sino que también permite registrar movimientos clave (auditoría) en la tabla historial\_stock, lo que representa una mejora en trazabilidad y control interno.
4. Funciones Definidas por el Usuario: Se diseñaron funciones para métricas operativas, como la disponibilidad de platos, el porcentaje de ocupación de mesas y la antigüedad de empleados. Estas funciones fortalecen el análisis interno del negocio y permiten ser integradas en dashboards o reportes automatizados.
5. Optimización del Rendimiento: El análisis con EXPLAIN, SHOW PROFILES y pruebas de carga (>500 registros) permitió detectar cuellos de botella y aplicar mejoras mediante índices simples y compuestos estratégicamente ubicados. Los tiempos de respuesta mejoraron significativamente en consultas comunes tras su implementación.
6. Seguridad y Calidad de Datos: Se utilizaron restricciones como UNIQUE, NOT NULL, CHECK y relaciones ON DELETE/ON UPDATE que garantizan integridad y minimizan errores humanos. Se previó además la validación de datos críticos a través de triggers, procedimientos y estructuras auxiliares (catálogos).
7. Preparación para Producción y Escalabilidad: El sistema se encuentra preparado para su implementación en un entorno productivo, gracias a la documentación clara, el uso de convenciones técnicas, y la previsión de crecimiento. La estructura es compatible con herramientas de reporting, visualización y futuras integraciones web.

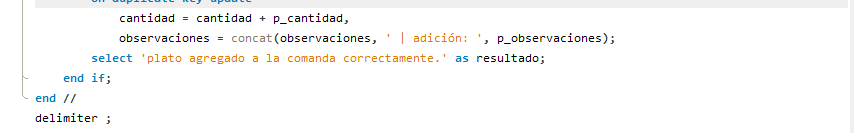
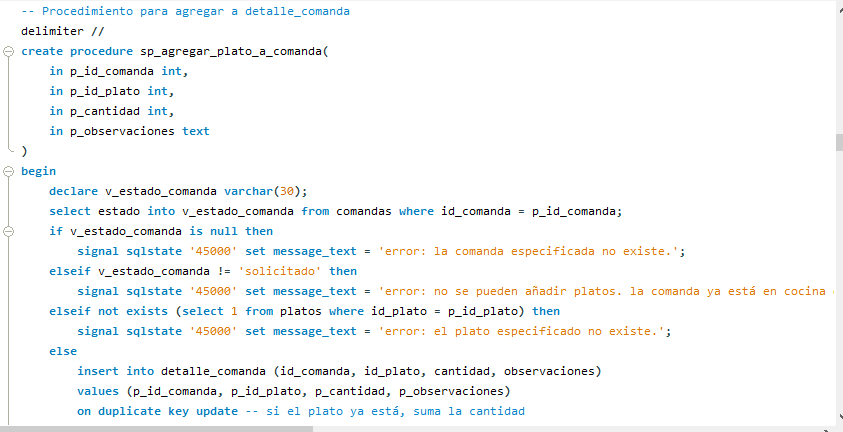
### 13. Recomendaciones

1. Revisar periódicamente los índices implementados: Tras haber creado índices simples y compuestos estratégicos en tablas como platos, facturas, detalle\_comanda o comandas, recomiendo analizarlos cada cierto tiempo usando EXPLAIN o herramientas gráficas, especialmente si cambian las consultas más frecuentes o aumenta el volumen de datos.
2. Evitar la sobreindexación innecesaria: A pesar de que los índices mejoran el rendimiento en consultas SELECT, también pueden afectar negativamente la velocidad de inserciones y actualizaciones. Por ello, mantendré solo los índices que realmente sean útiles en términos de acceso y frecuencia de uso.
3. Aplicar mantenimiento y optimización periódica de tablas: Es recomendable usar comandos como OPTIMIZE TABLE en las tablas más activas (por ejemplo, detalle\_comanda, historial\_stock, facturas), para evitar fragmentación y mantener el rendimiento óptimo del motor.
4. Continuar fortaleciendo la lógica de negocio con procedimientos almacenados y triggers: La automatización lograda en el control de turnos, la actualización de stock, el cálculo de importes, y la validación de reglas internas ha demostrado ser efectiva. Recomiendo seguir este enfoque para nuevas reglas o necesidades operativas.
5. Establecer políticas de respaldo (backups) y restauración: A medida que el sistema entre en uso real o semiproducción, será clave programar respaldos periódicos automáticos y probar los procesos de restauración, garantizando la seguridad y continuidad operativa de los datos.
6. Crear vistas y capas de seguridad lógica: Para consultas específicas o futuras integraciones externas (como aplicaciones web o móviles), propongo el uso de vistas SQL que oculten columnas sensibles y expongan solo los datos necesarios, mejorando el control de acceso.
7. Monitorear el crecimiento de datos en tablas críticas: Algunas tablas como facturas, detalle\_comanda y historial\_stock tienen potencial de crecer rápidamente. Si el volumen se vuelve muy alto, podría considerar técnicas como archivado de datos antiguos, o incluso particionado de tablas.
8. Aprovechar las funciones analíticas creadas: Las funciones que implementé para métricas como platos disponibles, ocupación de mesas o antigüedad de empleados, pueden utilizarse como base para construir reportes gerenciales o dashboards. Recomiendo integrarlas en herramientas de análisis o interfaces web en versiones futuras.

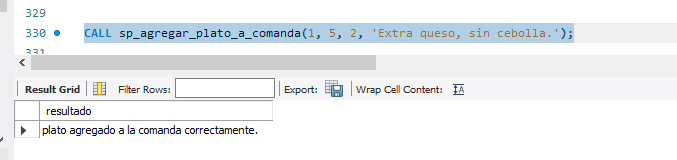
### 14.Anexos

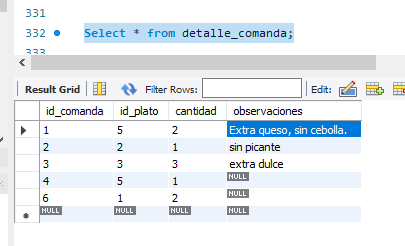
**Validación Procedimientos almacenados:**

Procedimiento 1:

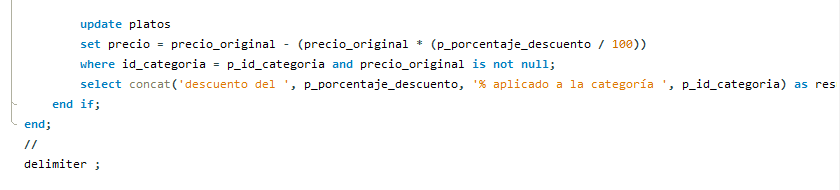
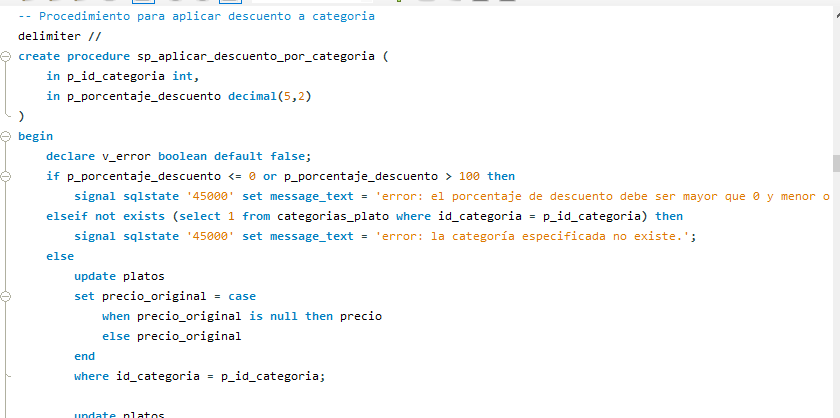


Validación:

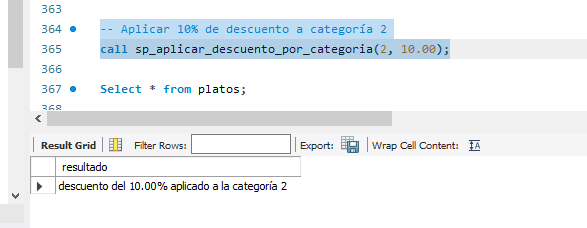


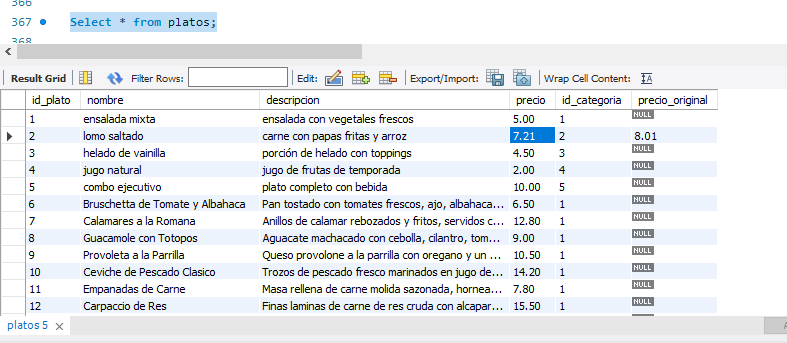


Procedimiento 2:

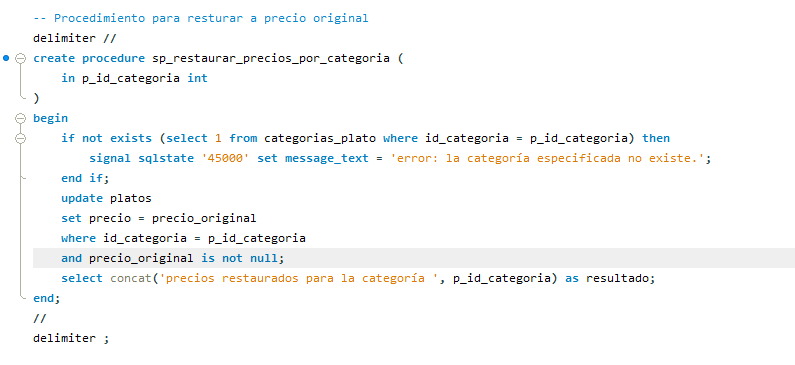


Validación:

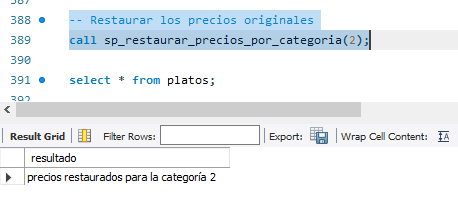


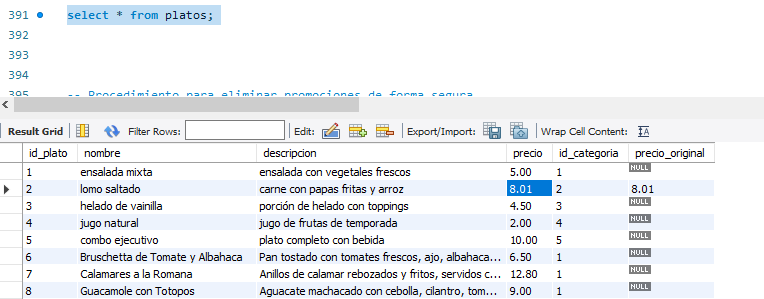


Procedimiento 3:

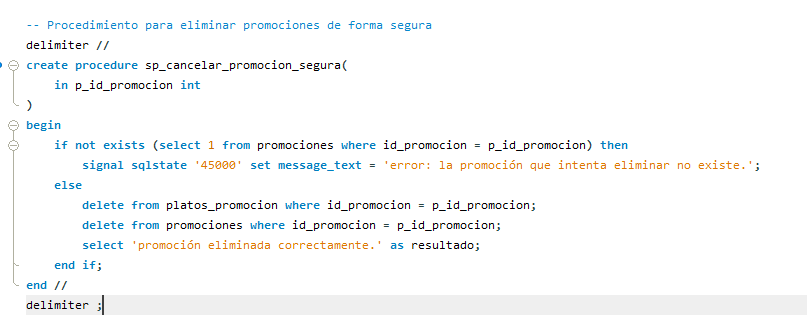


Validación:

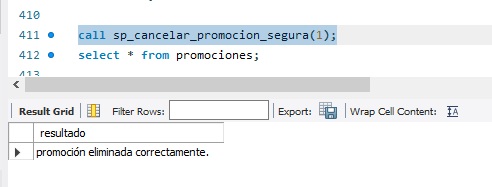


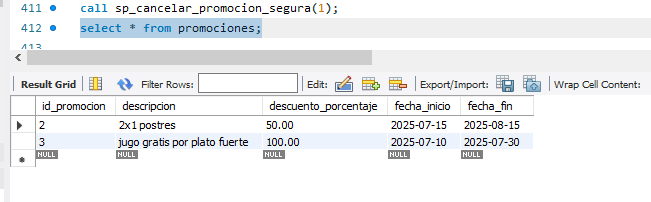


Procedimiento 4:

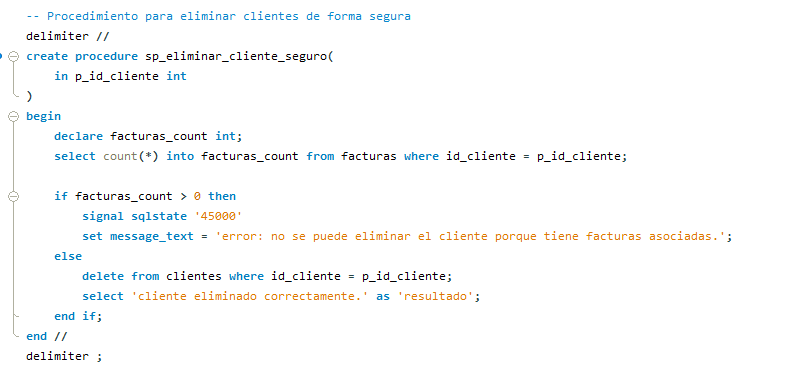


Validación:

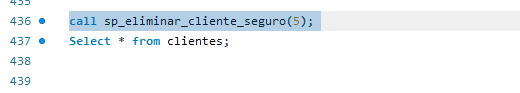




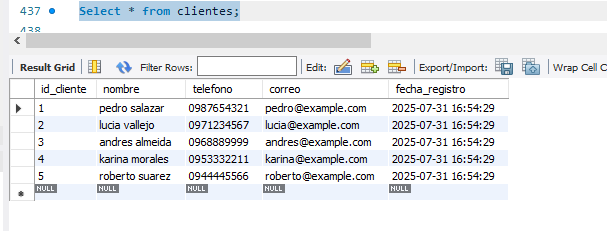
Procedimiento 5:



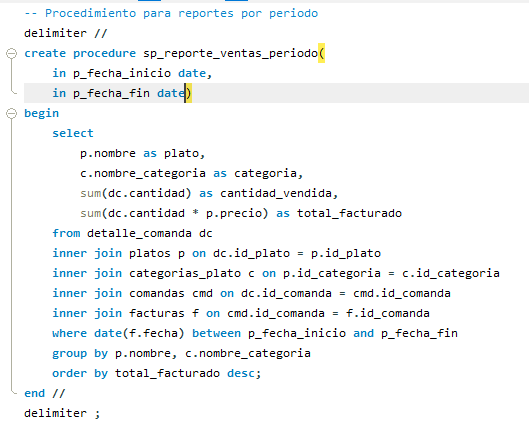
Validación:



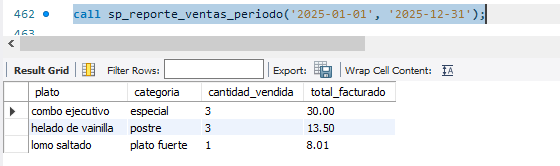




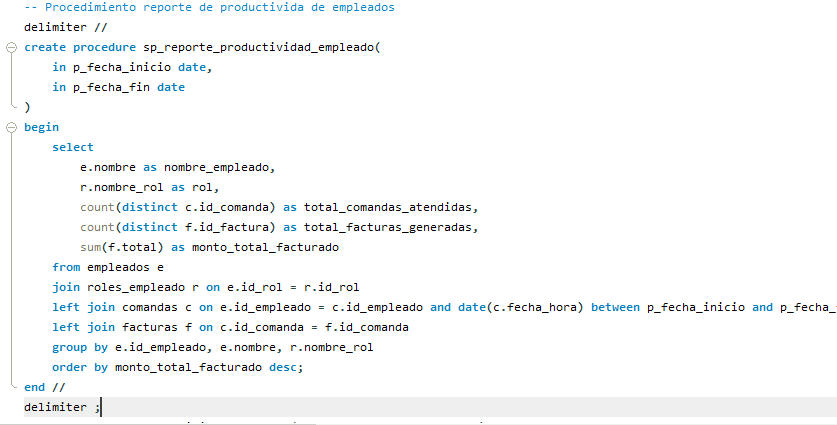
Procedimiento 6:



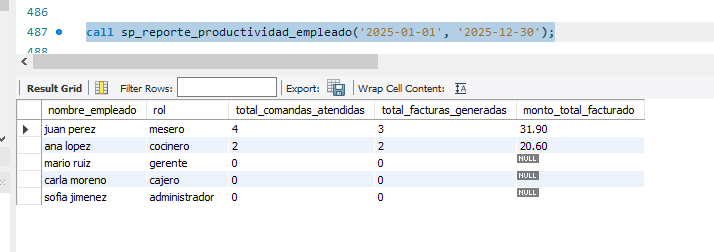
Validación:



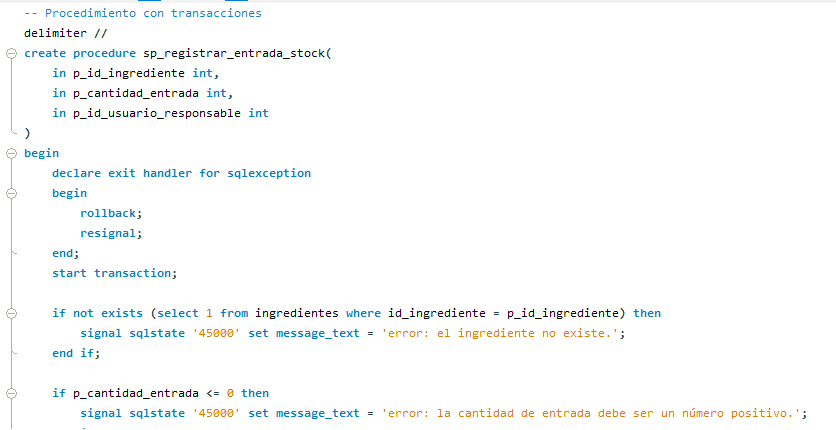
Procedimiento 7:

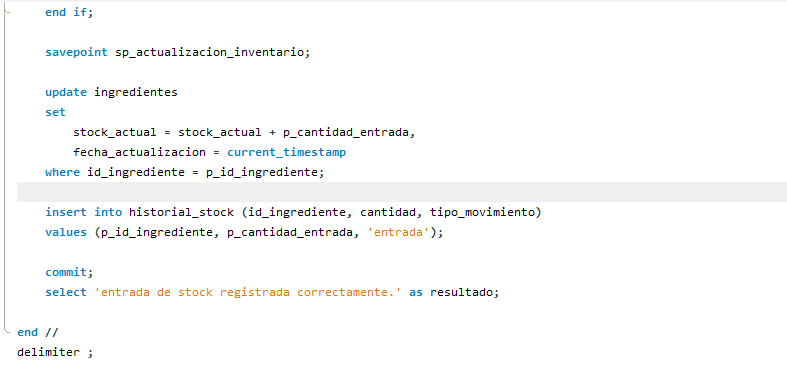


Validación:

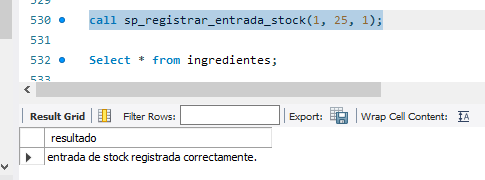


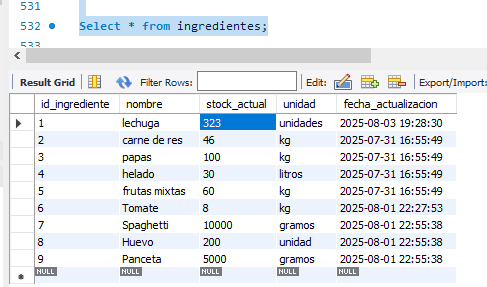
Procedimiento 8:

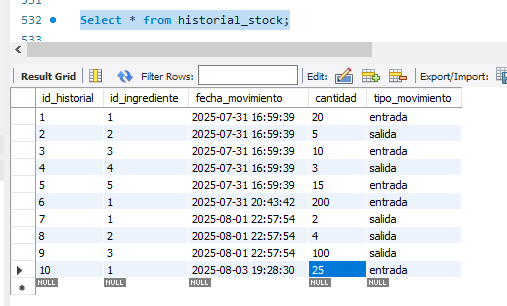




Validación:

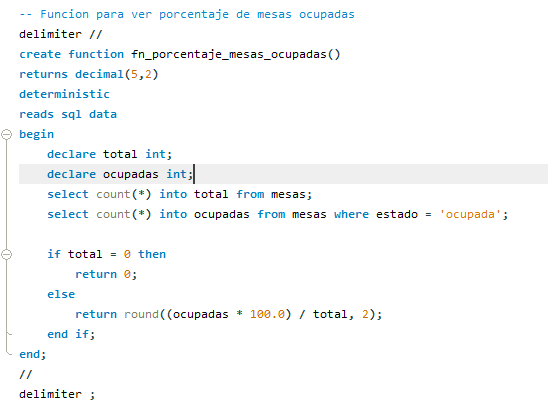




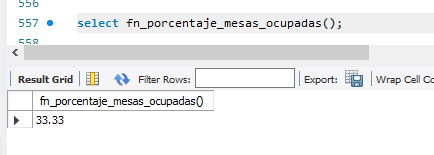


**Validación Funciones:**

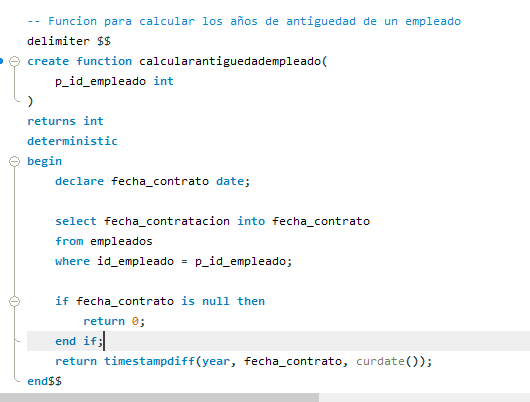
Función 1:



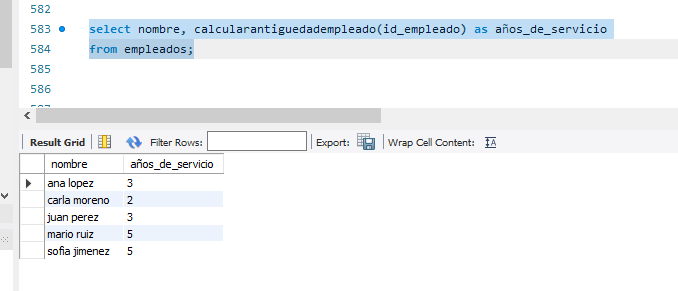
Validación:



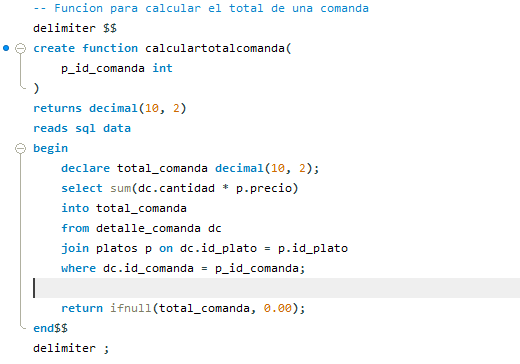
Función 2:



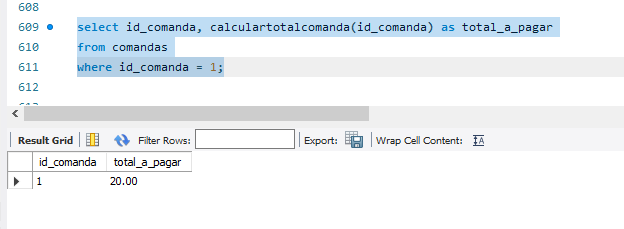
Validación:



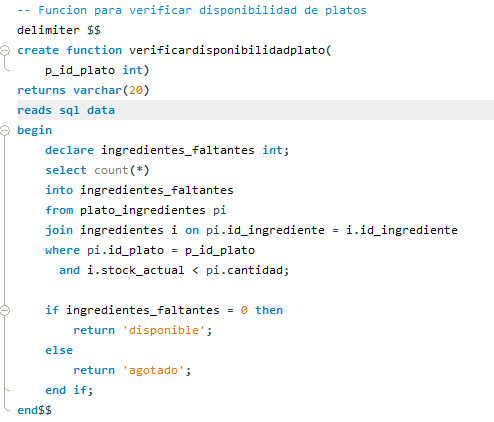
Función 3:



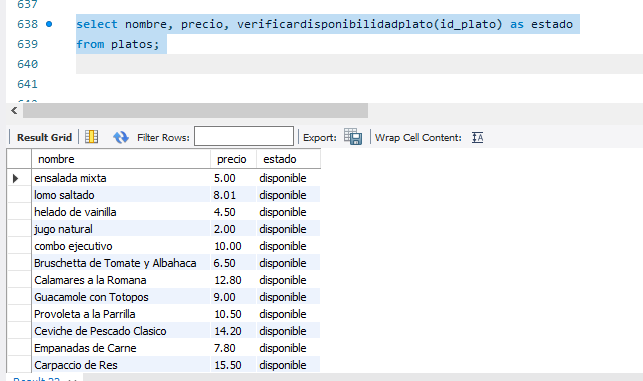
Validación:



Función 4:

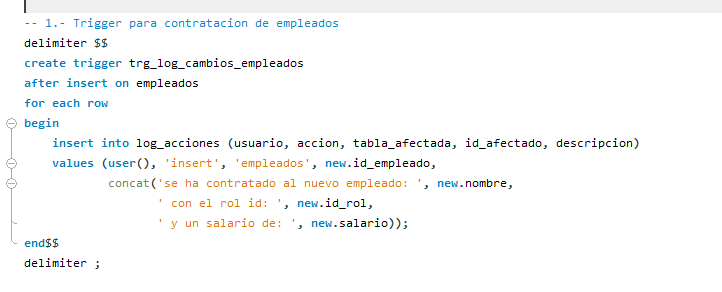


Validacion:

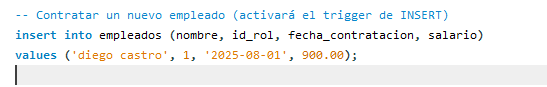


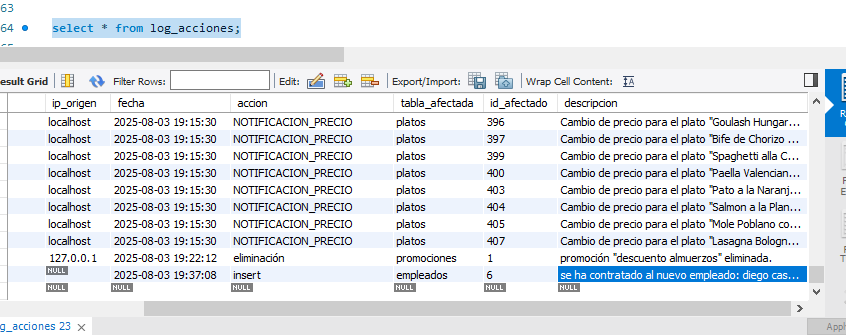
Validación Triggers:

Trigger 1:

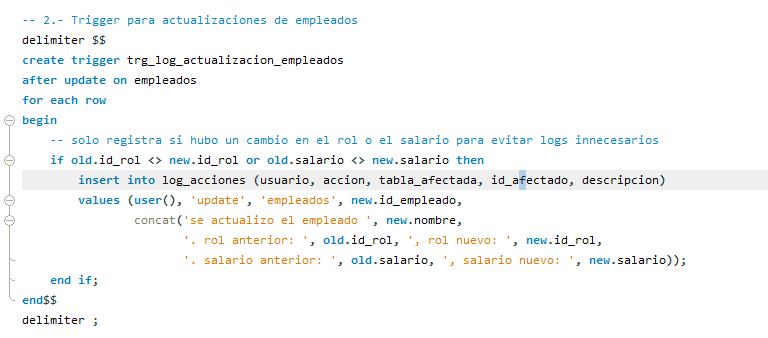


Validación:

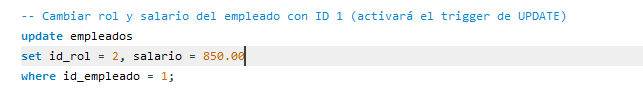


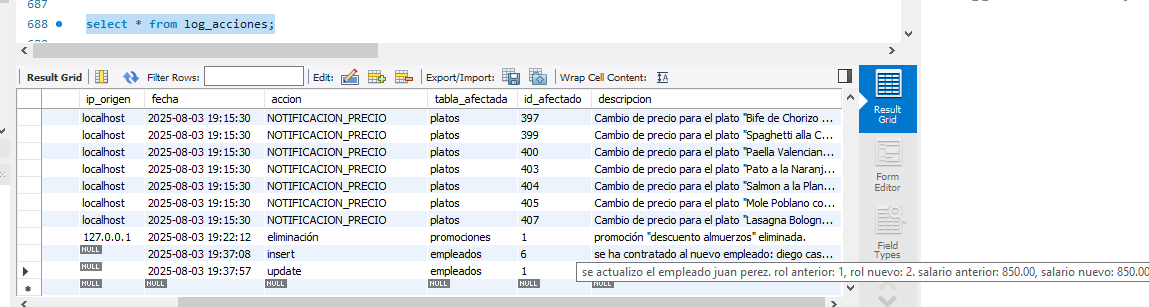


Trigger 2:

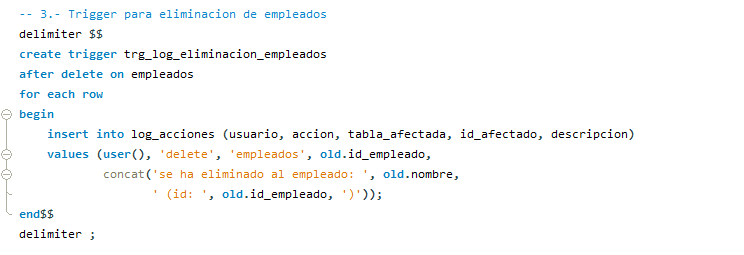


Validación:

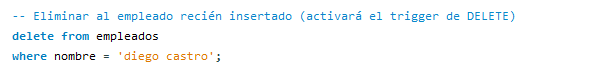


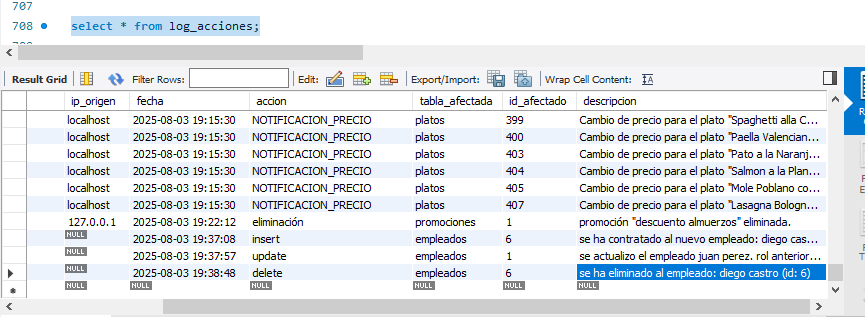


Trigger 3:

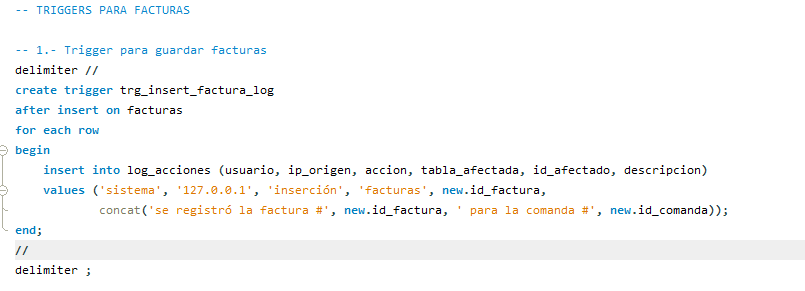


Validación:

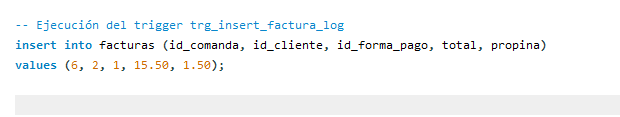


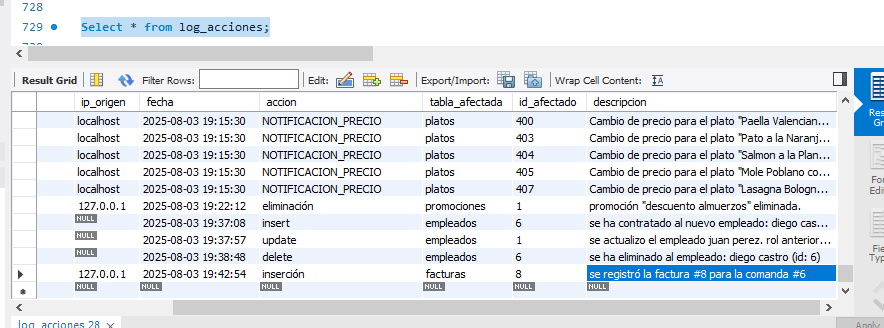


Trigger4:

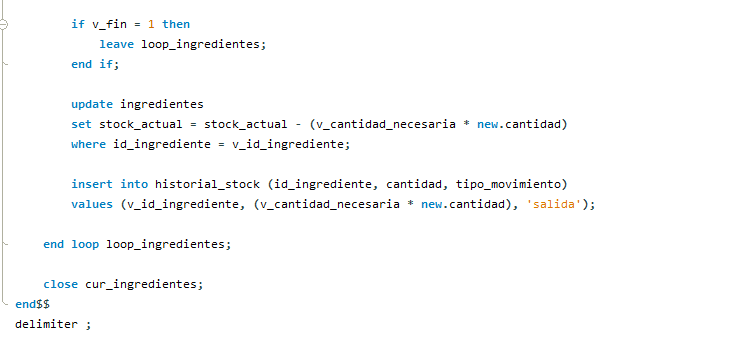
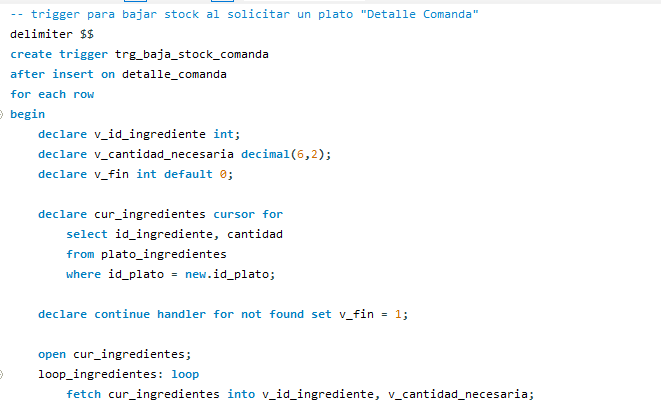


Validación:



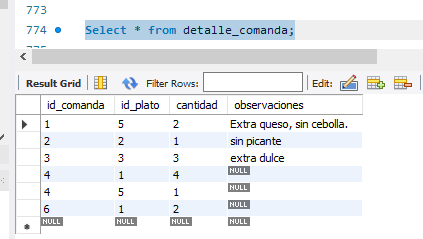


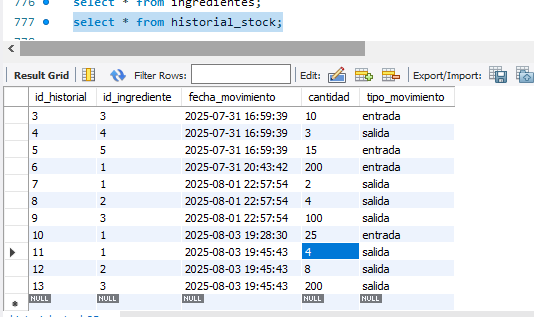
Trigger 5:



Validación:



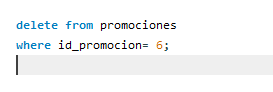


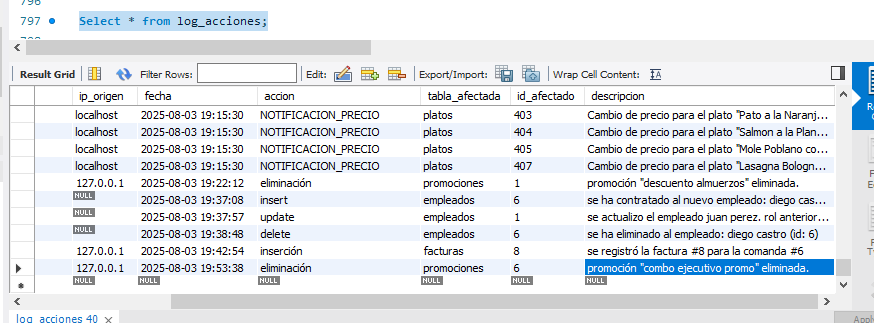


Trigger 6:

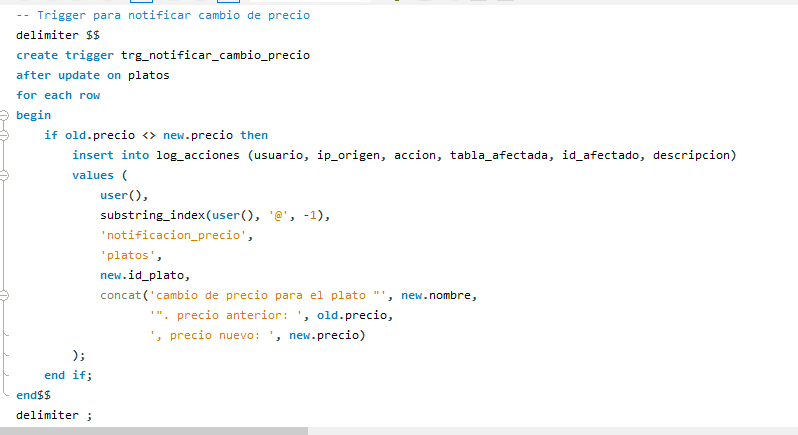


Validación:

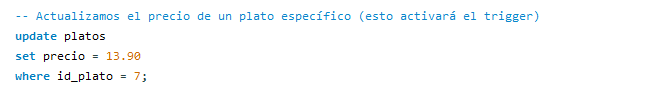


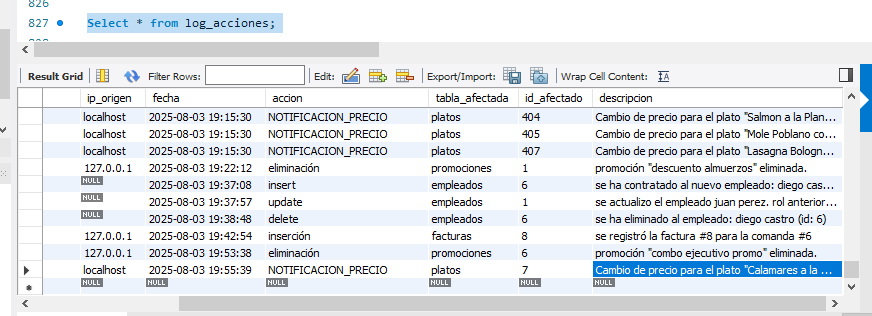


Trigger 7:



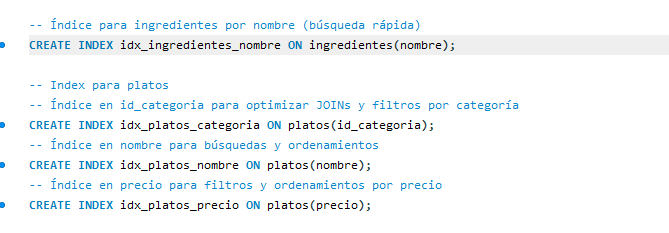
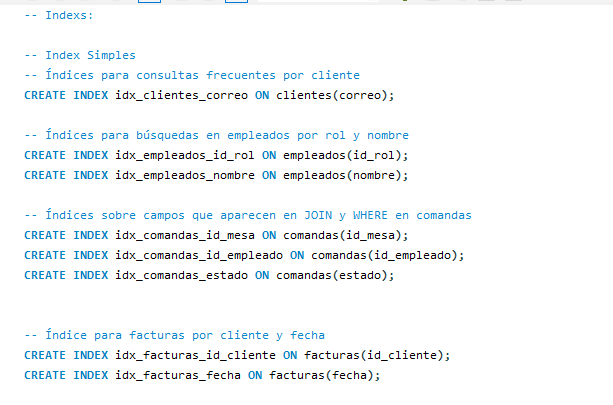
Validación:



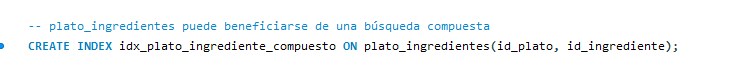
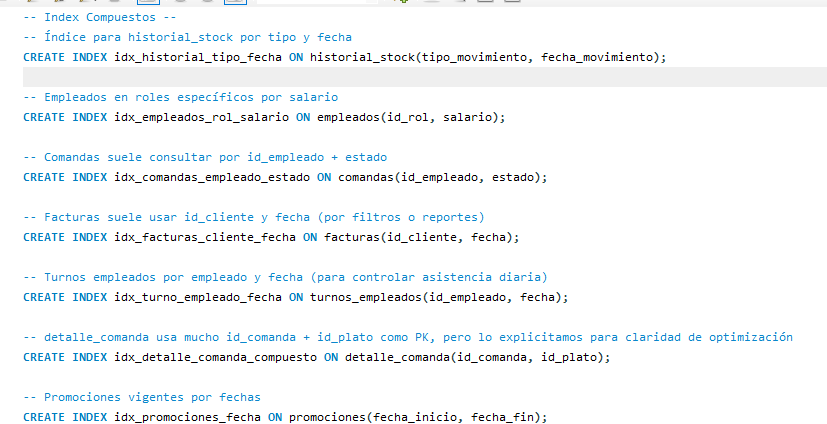


Validación Index:

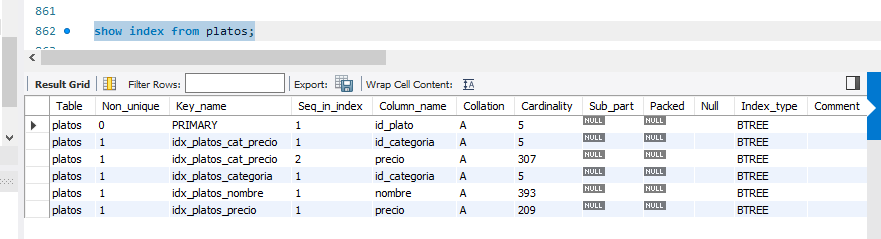
Simples:

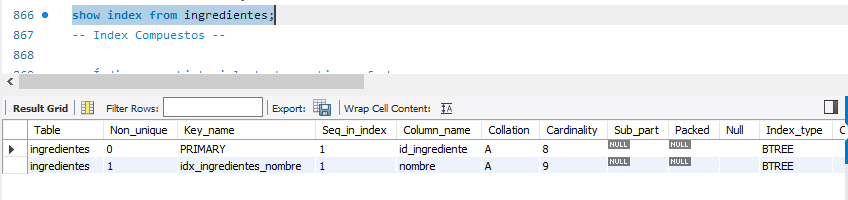
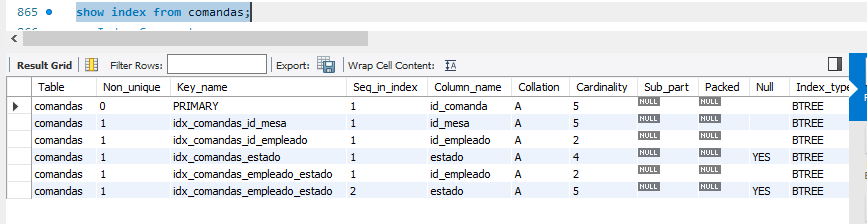
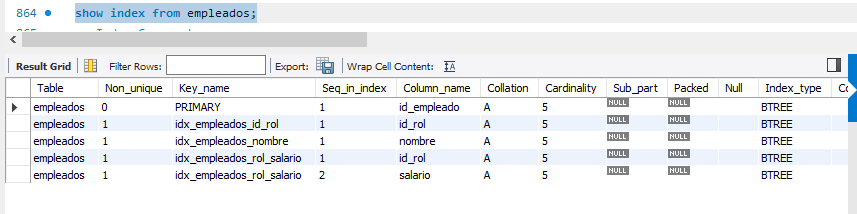
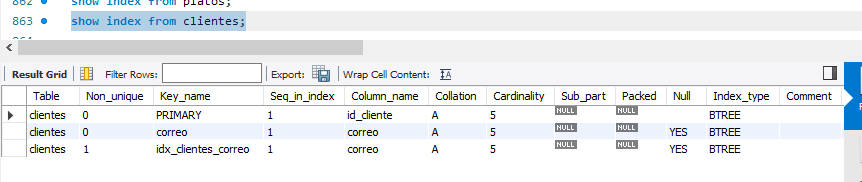


Compuestos:

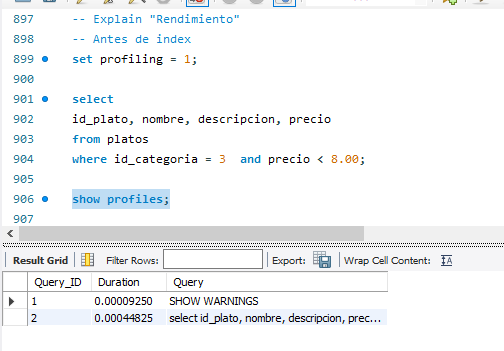


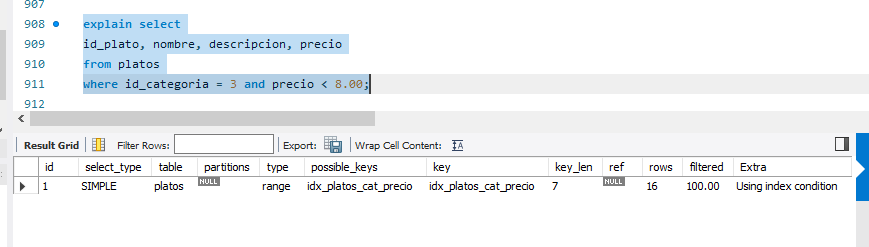
Validación:

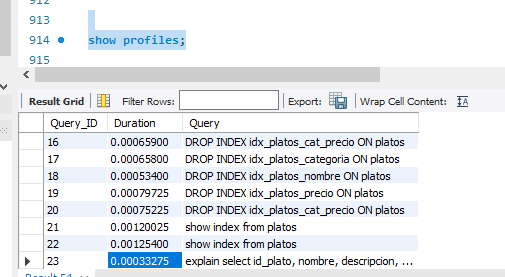


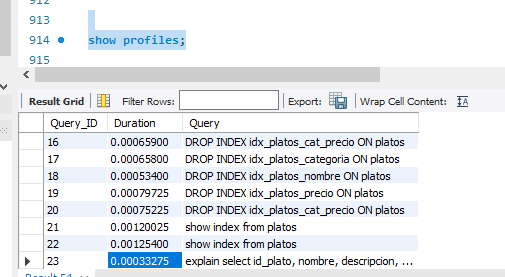


Validación Rendimiento “Explain”

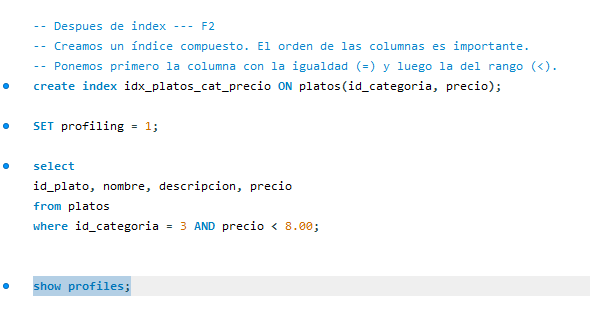


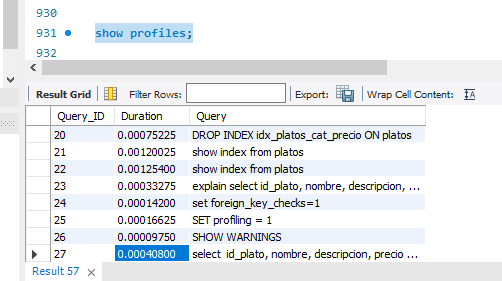


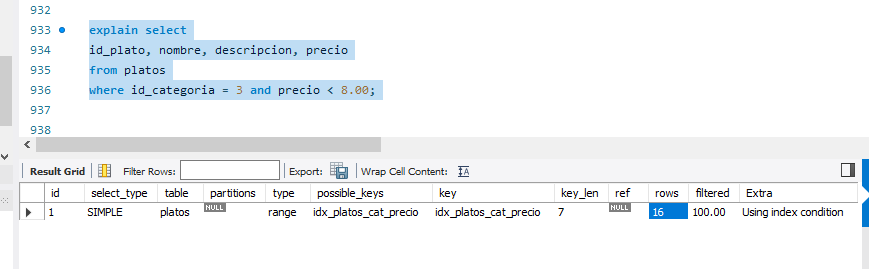




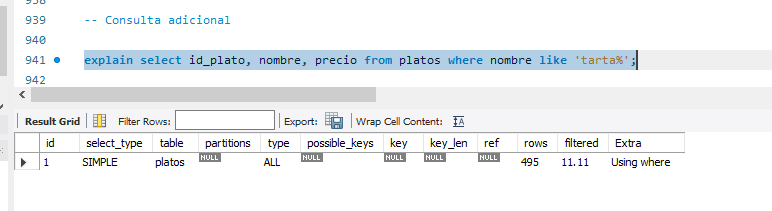
Después de crear index:







Consulta adicional:



Después de crear index:

