

SISTEMA INTEGRAL DE GESTIÓN PARA ENTORNOS PROFESIONALES CON BASES DE DATOS RELACIONALES

SISTEMA DE GESTIÓN HOSPITALARIA

Autores: Nicolás Chiguano
Melany Perugachi
Fecha: Agosto 2025

Tabla de contenido

1. Introducción	3
2. Objetivo General	3
3. Objetivos específico.....	3
4. Descripción General del Sistema	3
5. Componentes Técnicos.....	3
5.1 Modelo de Datos y Normalización.....	3
5.2 Procedimientos Almacenados	4
5.3 Funciones y Triggers	12
5.3.1 Funciones	12
5.3.2 Triggers.....	15
5.3.3 Consultas de control	20
5.4 Índices y Optimización	22
5.4.1 Índices Simples	22
5.4.2 Índices compuestos	24
5.4.3 Simular carga con 500+ registros.....	28
5.4.3.1 Anexos	32
5.5 Seguridad y Roles	34
5.5.1 Roles	34
5.5.2 USUARIOS Y ASIGNACIÓN DE ROLES	34
5.5.3 Rol por defecto	34
5.5.4 Asignación de privilegios	35
5.5.5 REVOCACIÓN DE PRIVILEGIOS CON REVOKE	35
5.5.5 ENCRIPCIÓN DEMOSTRATIVA.....	35
-- Hash con SHA2 y MD5	35
5.5.6 Cifrado y descifrado simétrico con AES	35
5.5.7 VALIDACIÓN DE ENTRADAS CON REGEXP	36
5.5.8 SIMULACIÓN DE INTENTOS FALLIDOS.....	36
5.6 Auditoría.....	37
5.7 Respaldo y Recuperación.....	38
6. Resultados y Pruebas.....	44
7. Conclusiones.....	44



1. Introducción

Este informe describe el desarrollo de un sistema de simulación para un entorno de gestión hospitalaria profesional. El propósito principal es usar conocimientos sobre modelos, seguridad, funcionamiento, pasos a seguir, tareas, auditoría y papeles en un entorno básico de datos relacional con MySQL.

2. Objetivo General

Diseñar e implementar un sistema de gestión hospitalaria basado en MySQL que gestione pacientes, citas, historia clínica, recetas y facturación, asegurando la integridad referencial, la seguridad de los datos, la eficiencia en las consultas, la copia de seguridad automatizada y la auditoría, apegándose a los estándares profesionales para entornos del mundo real.

3. Objetivos específico

- Desarrollar procedimientos, funciones y triggers que automaticen operaciones clave del sistema, garanticen la integridad de los datos y simulen el comportamiento profesional de una base de datos.

4. Descripción General del Sistema

El sistema que desarrollamos está pensado para gestionar de forma ordenada y segura la información de un hospital. Tomamos en cuenta y nos enfocamos en automatizar procesos como el registro de pacientes, agendamiento, control de historial clínico, recetas y facturación. También incluye roles diferenciados, auditoría de acciones, encriptación, triggers automáticos, seguridad todo esto lo realizamos en MySQL.

5. Componentes Técnicos

5.1 Modelo de Datos y Normalización

El modelo incluye más de 12 tablas normalizadas en tercera forma normal (3FN), con relaciones uno a muchos y muchos a muchos. Se implementan claves foráneas, restricciones 'NOT NULL', 'UNIQUE', 'DEFAULT', 'AUTO_INCREMENT' y catálogos independientes.

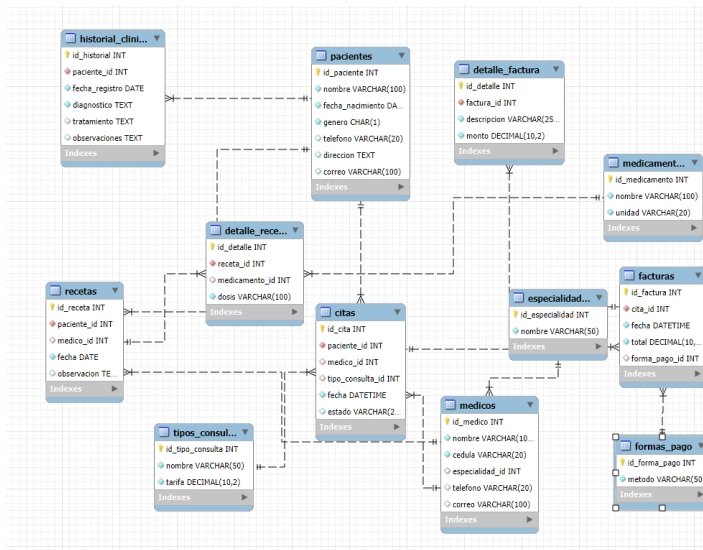


Figure 1

5.2 Procedimientos Almacenados

Se desarrollaron más de 10 procedimientos que cubren inserciones con validación cruzada, actualizaciones masivas, eliminaciones seguras, facturación automática individual y en lote, y reportes por período. Todos usan transacciones y manejo de errores.

1. sp_insertar_paciente_con_validacion

¿Qué hace?

Inserta un nuevo paciente en la base de datos solo si no existe otro con el mismo correo electrónico. Esta verificación ayuda a evitar duplicados y mantener la integridad de los datos de contacto.

Tipo: Inserción con validación cruzada.

Útil para: Validar unicidad antes de guardar información sensible como correos.

```

3  -- 1. Registrar una cita con validación de paciente y médico
4  DELIMITER $$
5  ● CREATE PROCEDURE sp_registrar_cita (
6      IN p_id_paciente INT,
7      IN p_id_medico INT,
8      IN p_id_tipo_consulta INT,
9      IN p_fecha DATETIME
10 )
11 BEGIN
12     DECLARE existe_paciente INT DEFAULT 0;
13     DECLARE existe_medico INT DEFAULT 0;
14
15     SELECT COUNT(*) INTO existe_paciente FROM pacientes WHERE id_paciente = p_id_paciente;
16     IF existe_paciente = 0 THEN
17         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El paciente no existe.';
18     END IF;
19
20     SELECT COUNT(*) INTO existe_medico FROM medicos WHERE id_medico = p_id_medico;
21     IF existe_medico = 0 THEN
22         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El médico no existe.';
23     END IF;
24
25     INSERT INTO citas (paciente_id, medico_id, tipo_consulta_id, fecha, estado)
26     VALUES (p_id_paciente, p_id_medico, p_id_tipo_consulta, p_fecha, 'pendiente');
27 END $$
28 DELIMITER ;
29 ● CALL sp_registrar_cita(1, 1, 1, '2025-08-01 09:00:00');
30 ● select * from citas;
31

```

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

	id_cita	paciente_id	medico_id	tipo_consulta_id	fecha	estado
▶	1	1	1	1	2025-07-01 09:00:00	pendiente
	2	2	2	2	2025-07-02 10:00:00	pendiente
	3	3	3	3	2025-07-03 11:00:00	confirmada
	4	4	4	4	2025-07-04 12:00:00	cancelada
	5	5	5	5	2025-07-05 13:00:00	pendiente
	6	6	6	6	2025-07-06 14:00:00	confirmada
	7	7	7	7	2025-07-07 15:00:00	pendiente

citas 1

Apply

Revert

Result Grid

Form Editor

Figure 2

2. sp_actualizar_citas_pendientes (versión corregida)

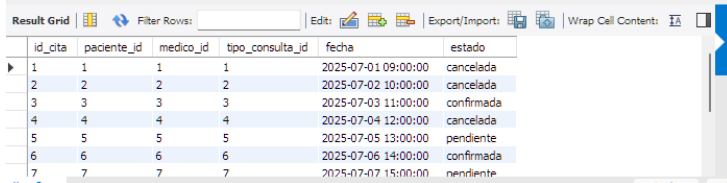
¿Qué hace?

Actualiza el estado de las citas a 'cancelada', 'realizada' u otro estado, a partir de una fecha específica, pero solo si están actualmente en estado 'pendiente'.

Tipo: Actualización masiva por condición.

Evita errores con una cláusula WHERE bien definida para cumplir con el modo seguro de MySQL.

```
32 -- 2. Actualizar estado de citas pendientes antes de una fecha dada.
33 DELIMITER $$
34 • CREATE PROCEDURE sp_actualizar_citas_pendientes (
35     IN p_fecha_limite DATETIME,
36     IN p_nuevo_estado VARCHAR(20)
37 )
38 BEGIN
39     UPDATE citas
40     SET estado = p_nuevo_estado
41     WHERE id_cita IN (
42         SELECT id_cita
43         FROM (
44             SELECT id_cita
45             FROM citas
46             WHERE estado = 'pendiente' AND fecha < p_fecha_limite
47         ) AS subconsulta_segura
48     );
49 END $$
50 DELIMITER ;
51
52 • CALL sp_actualizar_citas_pendientes('2025-07-05 00:00:00', 'cancelada');
53 • select * from citas;
54
55 -- 3. Eliminación segura de un paciente, solo si no tiene citas pendientes
```



	id_cita	paciente_id	medico_id	tipo_consulta_id	fecha	estado
1	1	1	1	1	2025-07-01 09:00:00	cancelada
2	2	2	2	2	2025-07-02 10:00:00	cancelada
3	3	3	3	3	2025-07-03 11:00:00	confirmada
4	4	4	4	4	2025-07-04 12:00:00	cancelada
5	5	5	5	5	2025-07-05 13:00:00	pendiente
6	6	6	6	6	2025-07-06 14:00:00	confirmada
7	7	7	7	7	2025-07-07 15:00:00	pendiente

Figure 3

3. sp_eliminar_paciente_seguro

¿Qué hace?

Intenta eliminar un paciente, pero solo si no tiene citas registradas. Si existen citas asociadas, no realiza la eliminación para no romper relaciones con otras tablas.

Tipo: Eliminación segura.

Beneficio: Protege la base de datos contra errores por violaciones de claves foráneas.

```

55 -- 3. Eliminación segura de un paciente, solo si no tiene citas pendientes.
56 DELIMITER $$
57 • CREATE PROCEDURE sp_eliminar_paciente_seguro (
58     IN p_id_paciente INT
59 )
60 BEGIN
61     DECLARE citas_pendientes INT DEFAULT 0;
62
63     SELECT COUNT(*) INTO citas_pendientes
64     FROM citas
65     WHERE paciente_id = p_id_paciente AND estado = 'pendiente';
66
67     IF citas_pendientes > 0 THEN
68         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede eliminar paciente con citas pendientes.';
69     ELSE
70         DELETE FROM pacientes WHERE id_paciente = p_id_paciente;
71     END IF;
72 END $$
73 DELIMITER ;
74
75 • CALL sp_eliminar_paciente_seguro(2);
76 • select * from pacientes;
77

```

id_paciente	nombre	fecha_nacimiento	genero	telefono	direccion	correo
1	Juan Pérez	1985-03-15	M	0998765432	Av. Siempre Viva 123	juan.perez@mail.com
3	Carlos Sánchez	1978-12-05	M	0976543210	Av. Central 789	carlos.sanchez@mail.com
4	Ana Martínez	2000-01-30	F	0965432109	Calle Real 101	ana.martinez@mail.com
5	Luis Fernández	1995-06-12	M	0954321098	Av. Las Flores 202	luis.fernandez@mail.com
6	Sofía Torres	1988-09-25	F	0943210987	Calle Luna 303	sofia.torres@mail.com
7	Miguel Ruiz	1975-11-17	M	0932109876	Av. Sol 404	miguel.ruiz@mail.com
8	Laura Castillo	1997-04-08	F	0921098765	Calle Estrella 505	laura.castillo@mail.com

Figure 4

4. sp_reporte_citas_por_rango

¿Qué hace?

Muestra un informe completo de citas realizadas entre dos fechas, incluyendo datos del paciente, médico y tipo de consulta. Usa JOINS para presentar la información de manera clara y relacionada.

Tipo: Generación de reportes por período.

Ideal para: Revisar citas atendidas en semanas, meses o fechas específicas.

```

80 -- 4. Generar reporte de facturas por periodo cantidad y total.
81 DELIMITER $$
82 • CREATE PROCEDURE sp_reporte_facturas_por_periodo (
83     IN p_fecha_inicio DATETIME,
84     IN p_fecha_fin DATETIME
85 )
86 BEGIN
87     SELECT
88         COUNT(*) AS total_facturas,
89         SUM(total) AS monto_total
90     FROM facturas
91     WHERE fecha BETWEEN p_fecha_inicio AND p_fecha_fin;
92 END $$
93 DELIMITER ;
94
95 • CALL sp_reporte_facturas_por_periodo('2025-07-01 00:00:00', '2025-07-10 23:59:59');
96

```

total_facturas	monto_total
9	885.00

Figure 5

5. sp_facturar_cita

¿Qué hace?

Crea automáticamente una factura para una cita, utilizando la tarifa del tipo de consulta relacionada. La factura incluye la fecha actual y una forma de pago especificada.

Tipo: Facturación automática individual.

Beneficio: Agiliza el proceso de facturación sin intervención manual.

```
90
97 -- 5. Facturación automática para una cita transacción con manejo de errores
98 DELIMITER $$
99 CREATE PROCEDURE sp_facturar_cita (
100     IN p_id_cita INT,
101     IN p_id_forma_pago INT
102 )
103 BEGIN
104     DECLARE v_total DECIMAL(10,2) DEFAULT 0;
105     DECLARE CONTINUE HANDLER FOR SQLSTATE '45000' SET MESSAGE_TEXT = 'Error durante facturación, se deshizo la transacción.';
106     BEGIN
107         ROLLBACK;
108         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error durante facturación, se deshizo la transacción.';
109     END;
110     START TRANSACTION;
111     SELECT cc.tarifa INTO v_total
112     FROM citas c
113     JOIN tipos_consulta tc ON c.tipo_consulta_id = tc.id_tipo_consulta
114     WHERE c.id_cita = p_id_cita;
115     INSERT INTO facturas (cita_id, fecha, total, forma_pago_id)
116     VALUES (p_id_cita, NOW(), v_total, p_id_forma_pago);
117     INSERT INTO detalle_factura (factura_id, descripcion, monto)
118     VALUES (LAST_INSERT_ID(), 'Tarifa consulta', v_total);
119     COMMIT;
120 END $$
121 DELIMITER |
122 CALL sp_facturar_cita(3, 1);
123 select * from facturas;
124 select * from detalle_factura;
```

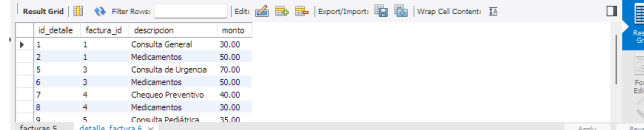


Figure 6

6. sp_actualizar_tarifas_porcentaje

¿Qué hace?

Aumenta (o disminuye) todas las tarifas de consulta en un porcentaje definido. Por ejemplo, puede incrementar todos los precios un 10%.

Tipo: Actualización masiva.

Útil para: Ajustes de tarifas por inflación, cambios anuales o promociones.

```

126 -- 6. Insertar medicamento con validación de nombre único
127 DELIMITER $$
128 CREATE PROCEDURE sp_insertar_medicamento (
129     IN p_nombre VARCHAR(100),
130     IN p_unidad VARCHAR(20)
131 )
132 BEGIN
133     DECLARE existe_medicamento INT DEFAULT 0;
134
135     SELECT COUNT(*) INTO existe_medicamento FROM medicamentos WHERE nombre = p_nombre;
136     IF existe_medicamento > 0 THEN
137         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El medicamento ya existe.';
138     ELSE
139         INSERT INTO medicamentos(nombre, unidad) VALUES (p_nombre, p_unidad);
140     END IF;
141 END $$
142 DELIMITER ;
143
144 CALL sp_insertar_medicamento('Diclofenaco', 'mg');
145 select * from medicamentos;
146

```

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:			
	id_medicamento	nombre	unidad
▶	1	Paracetamol	mg
	2	Ibuprofeno	mg
	3	Amoxicilina	mg
	4	Metformina	mg
	5	Omeprazol	mg
	6	Loratadina	mg
	7	Aspirina	mn

medicamentos 7 x

Figure 7

7. sp_actualizar_tarifa_por_id

¿Qué hace?

Modifica la tarifa de un tipo de consulta específico según su ID. Aumenta (o disminuye) la tarifa en un porcentaje dado.

Tipo: Actualización específica por ID.

Ventaja: Permite ajustes individuales sin afectar otras tarifas.

```

149 -- 7. Actualizar tarifas de tipos de consulta por porcentaje
150 DELIMITER $$
151 CREATE PROCEDURE actualizar_tarifa(
152     IN p_id_tipo INT,
153     IN p_porcentaje DECIMAL(5,2)
154 )
155 BEGIN
156     UPDATE tipos_consulta
157     SET tarifa = tarifa + (tarifa * p_porcentaje / 100)
158     WHERE id_tipo_consulta = p_id_tipo;
159 END $$
160 DELIMITER ;
161
162 CALL actualizar_tarifa(2, 10);
163 select * from tipos_consulta;

```

id_tipo_consulta	nombre	tarifa
1	Consulta General	30.00
2	Consulta Especializada	55.00
3	Consulta de Urgencia	70.00
4	Chequeo Preventivo	40.00
5	Consulta Pediátrica	35.00
6	Consulta Ginecológica	55.00
7	Consulta Neurológica	60.00

Figure 8

8. sp_registrar_historial_y_receta

¿Qué hace?

Registra simultáneamente un historial médico y su receta correspondiente. Usa una transacción para que, si ocurre un error en una de las inserciones, ambas operaciones se reviertan para evitar registros incompletos.

Tipo: Inserción con transacción controlada y manejo de errores.

Ventaja: Garantiza que los datos clínicos se guarden de forma segura y completa.

```

165 -- 8. Eliminación segura de médico sin citas pendientes
166 DELIMITER $$
167 CREATE PROCEDURE sp_eliminar_medico_seguro (
168     IN p_id_medico INT
169 )
170 BEGIN
171     DECLARE citas_pendientes INT DEFAULT 0;
172
173     SELECT COUNT(*) INTO citas_pendientes FROM citas
174     WHERE medico_id = p_id_medico AND estado = 'pendiente';
175
176     IF citas_pendientes > 0 THEN
177         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede eliminar médico con citas pendientes.';
178     ELSE
179         DELETE FROM medicos WHERE id_medico = p_id_medico;
180     END IF;
181 END $$
182 DELIMITER ;
183
184 CALL sp_eliminar_medico_seguro(4);
185 select * from medicos;

```

id_medico	nombre	cedula	especialidad_id	telefono	correo
1	Dr. Alberto Díaz	1234567890	1	0999999991	alberto.diaz@hospital.com
2	Dra. Patricia López	2345678901	2	0999999992	patricia.lopez@hospital.com
3	Dr. Ricardo Sánchez	3456789012	3	0999999993	ricardo.sanchez@hospital.com
5	Dr. Jorge Ramírez	5678901234	5	0999999995	jorge.ramirez@hospital.com
6	Dra. Marta Gutiérrez	6789012345	6	0999999996	marta.gutierrez@hospital.com
7	Dr. Luis Herrera	7890123456	7	0999999997	luis.herrera@hospital.com
8	Dra. Carmen Morales	8901234567	8	0999999998	carmen.morales@hospital.com

Figure 9

9. sp_eliminar_medico_seguro

¿Qué hace?

Elimina a un médico solo si no tiene citas asociadas. Si tiene citas registradas, no lo elimina y devuelve un mensaje informativo.

Tipo: Eliminación segura con validación previa.

Útil para: Mantener la consistencia referencial sin perder datos importantes.

```
189 -- 9. Reporte de pacientes registrados por fecha de nacimiento
190 DELIMITER $$
191 • CREATE PROCEDURE sp_reporte_pacientes_por_fecha (
192     IN p_fecha_inicio DATE,
193     IN p_fecha_fin DATE
194 )
195 BEGIN
196     SELECT id_paciente, nombre, fecha_nacimiento, genero, telefono, correo
197     FROM pacientes
198     WHERE fecha_nacimiento BETWEEN p_fecha_inicio AND p_fecha_fin
199     ORDER BY fecha_nacimiento;
200 END $$
201 DELIMITER ;
202
203 • CALL sp_reporte_pacientes_por_fecha('1980-01-01', '2000-12-31');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [FA](#)

	id_paciente	nombre	fecha_nacimiento	genero	telefono	correo
▶	9	Pedro Morales	1980-08-20	M	0910987654	pedro.morales@mail.com
1	Juan Pérez	1985-03-15	M	0998765432	juan.perez@mail.com	
6	Sofía Torres	1988-09-25	F	0943210987	sofia.torres@mail.com	
8	Laura Castillo	1992-04-08	F	0921098765	laura.castillo@mail.com	
5	Luis Fernández	1995-06-12	M	0954321098	luis.fernandez@mail.com	
10	Camila Ríos	1997-02-14	F	0909876543	camila.rios@mail.com	
4	Ana Martínez	2000-01-30	F	0965432109	ana.martinez@mail.com	

Result 10 x

Figure 10

10. sp_facturar_multiples_citas

¿Qué hace?

Recibe una lista de IDs de citas separadas por comas y genera una factura para cada una. Si alguna de las citas no es válida, la transacción completa se revierte, asegurando que no haya facturación parcial.

Tipo: Facturación automática en lote, con transacción y control de errores.

Ventaja: Eficiencia en el proceso de facturación masiva y control de integridad.

```

205 -- 10. Facturación múltiple de varias citas con transacción
206 DELIMITER $$
207 CREATE PROCEDURE sp_facturar_multiples_citas (
208     IN p_lista_citas TEXT,
209     IN p_id_forma_pago INT
210 )
211 BEGIN
212     DECLARE v_pos INT DEFAULT 1;
213     DECLARE v_longitud INT;
214     DECLARE v_id_cita INT;
215     DECLARE v_total DECIMAL(10,2);
216     DECLARE v_factura_id INT;
217     START TRANSACTION;
218     SET v_longitud = LENGTH(p_lista_citas) - LENGTH(REPLACE(p_lista_citas, ',', '')) + 1;
219
220     WHILE v_pos <= v_longitud DO
221         SET v_id_cita = CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(p_lista_citas, ',', v_pos), ',', -1) AS UNSIGNED);
222         SELECT tc.tarifa INTO v_total
223         FROM citas c
224         JOIN tipos_consulta tc ON c.tipo_consulta_id = tc.id_tipo_consulta
225         WHERE c.id_cita = v_id_cita;
226         INSERT INTO facturas (cita_id, fecha, total, forma_pago_id)
227         VALUES (v_id_cita, NOW(), v_total, p_id_forma_pago);
228         SET v_factura_id = LAST_INSERT_ID();
229         INSERT INTO detalle_factura (factura_id, descripcion, monto)
230         VALUES (v_factura_id, 'Tarifa consulta', v_total);
231         SET v_pos = v_pos + 1;
232     END WHILE;
233     COMMIT;
234 END $$
235 DELIMITER ;

```

```

237 CALL sp_facturar_multiples_citas('1,3', 1);
238 select * from detalle_factura;
239
240

```

Result Grid				
Filter Rows:				
	id_detalle	factura_id	descripcion	monto
	1	1	Consulta General	30.00
	2	1	Medicamentos	50.00
	5	3	Consulta de Urgencia	70.00
	6	3	Medicamentos	50.00
	7	4	Chequeo Preventivo	40.00
	8	4	Medicamentos	30.00
	9	5	Consulta Pediátrica	35.00
	10	5	Medicamentos	55.00
	11	11	Tarifa consulta	70.00
	12	12	Tarifa consulta	30.00
	13	13	Tarifa consulta	70.00
	* NULL	NULL	NULL	NULL

detalle factura 11

5.3 Funciones y Triggers

Las funciones calculan edad, porcentaje de citas canceladas y estado de riesgo clínico. Los triggers registran acciones clave, controlan automáticamente el stock y guardan historial de diagnósticos modificados.

5.3.1 Funciones

1. fn_calcular_edad

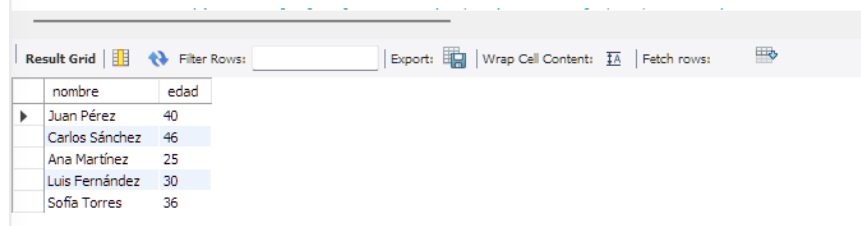
¿Qué hace?

Calcula la edad actual de un paciente a partir de su fecha de nacimiento.

Cómo funciona: Usa la función `TIMESTAMPDIFF` para calcular la diferencia en años entre la fecha de nacimiento y la fecha actual (`CURDATE()`).

Uso: Permite mostrar la edad exacta en años para reportes o validaciones.

```
241 -- FUNCIONES
242 -- 1. Función que calcula la edad de un paciente a partir de su fecha de nacimiento
243 DELIMITER $$
244 • CREATE FUNCTION fn_calcular_edad(fecha_nacimiento DATE)
245 RETURNS INT
246 DETERMINISTIC
247 BEGIN
248     DECLARE edad INT;
249     SET edad = TIMESTAMPDIFF(YEAR, fecha_nacimiento, CURDATE());
250     RETURN edad;
251 END $$
252 DELIMITER ;
253
254 • SELECT nombre, fn_calcular_edad(fecha_nacimiento) AS edad
255 FROM pacientes
256 LIMIT 5;
257
258
```



nombre	edad
Juan Pérez	40
Carlos Sánchez	46
Ana Martínez	25
Luis Fernández	30
Sofía Torres	36

Figure 11

2. fn_porcentaje_citas_canceladas

¿Qué hace?

Calcula el porcentaje de citas que un paciente ha cancelado en relación al total de citas que ha tenido.

Cómo funciona: Primero cuenta el total de citas del paciente, luego cuenta cuántas de esas citas están con estado 'cancelada'. Divide las canceladas por el total y multiplica por 100 para obtener el porcentaje. Si el paciente no tiene citas, devuelve 0.

Uso: Útil para evaluar la confiabilidad o compromiso del paciente con sus citas médicas.

```

3  -- 2. Función que calcula el porcentaje de citas canceladas de un paciente por su ID
4  DELIMITER $$
5  • CREATE FUNCTION fn_porcentaje_citas_canceladas(p_id_paciente INT)
6  RETURNS DECIMAL(5,2)
7  DETERMINISTIC
8  BEGIN
9      DECLARE total_citas INT DEFAULT 0;
10     DECLARE citas_canceladas INT DEFAULT 0;
11     DECLARE porcentaje DECIMAL(5,2) DEFAULT 0;
12
13     SELECT COUNT(*) INTO total_citas
14     FROM citas
15     WHERE paciente_id = p_id_paciente;
16
17     IF total_citas = 0 THEN
18         RETURN 0;
19     END IF;
20     SELECT COUNT(*) INTO citas_canceladas
21     FROM citas
22     WHERE paciente_id = p_id_paciente AND estado = 'cancelada';
23     SET porcentaje = (citas_canceladas / total_citas) * 100;
24
25     RETURN ROUND(porcentaje, 2);
26 END $$
27 DELIMITER ;
28
29 • SELECT nombre, fn_porcentaje_citas_canceladas(id_paciente) AS porcentaje_canceladas
30 FROM pacientes
31 LIMIT 5;

```

nombre	porcentaje_canceladas
Juan Pérez	50.00
Carlos Sánchez	0.00
Ana Martínez	100.00
Luis Fernández	0.00
Sofía Torres	0.00

Figure 12

3. fn_estado_riesgo

¿Qué hace?

Evalúa el estado de riesgo de un paciente basado en la cantidad de diagnósticos “graves” registrados en su historial clínico.

Cómo funciona: Cuenta cuántos registros en el historial tienen la palabra "grave" en el diagnóstico.

- Si tiene 3 o más, retorna 'Alto riesgo'.
- Si tiene 1 o 2, retorna 'Riesgo moderado'.
- Si no tiene ninguno, retorna 'Sin riesgo'.

Uso: Proporciona una evaluación rápida y simple del riesgo médico para priorizar atención.

```

288 -- 3. Función que devuelve el estado de riesgo del paciente basado en
289 -- la cantidad de citas con diagnóstico grave ejemplo simple
290 DELIMITER $$
291 • CREATE FUNCTION fn_estado_riesgo(p_id_paciente INT)
292 RETURNS VARCHAR(20)
293 DETERMINISTIC
294 BEGIN
295     DECLARE cantidad_graves INT DEFAULT 0;
296
297     SELECT COUNT(*) INTO cantidad_graves
298     FROM historial_clinico
299     WHERE paciente_id = p_id_paciente AND diagnostico LIKE '%grave%';
300
301     IF cantidad_graves >= 3 THEN
302         RETURN 'Alto riesgo';
303     ELSEIF cantidad_graves BETWEEN 1 AND 2 THEN
304         RETURN 'Riesgo moderado';
305     ELSE
306         RETURN 'Sin riesgo';
307     END IF;
308 END $$
309 DELIMITER ;
310
311 • SELECT nombre, fn_estado_riesgo(id_paciente) AS estado_riesgo
312 FROM pacientes
313 LIMIT 5;
314
315

```

Result Grid		Filter Rows:	Export:	Wrap Cell Contents:	Fetch rows:
nombre	estado_riesgo				
Juan Pérez	Sin riesgo				
Carlos Sánchez	Sin riesgo				
Ana Martínez	Sin riesgo				
Luis Fernández	Sin riesgo				
Sofía Torres	Sin riesgo				

Figure 13

5.3.2 Triggers

1. tr_auditoria_delete_paciente:

¿Qué hace:?

Este trigger se activa después de eliminar un paciente.

Cómo funciona: Su función es registrar en la tabla log_acciones la información del usuario que realizó la acción, la IP, el rol, el nombre de la tabla afectada y el ID del paciente eliminado. Esto permite mantener un historial de auditoría de las eliminaciones realizadas.


```

315 -- TRIGGERS
316 -- Tabla y trigger para eliminar paciente
317 CREATE TABLE log_acciones (
318     id_log INT AUTO_INCREMENT PRIMARY KEY,
319     usuario VARCHAR(50),
320     ip_cliente VARCHAR(45),
321     terminal VARCHAR(50),
322     rol_activo VARCHAR(50),
323     accion VARCHAR(100),
324     tabla VARCHAR(50),
325     id_afectado INT,
326     transaccion TEXT,
327     fecha DATETIME DEFAULT CURRENT_TIMESTAMP
328 );
329
330 DELIMITER $$
331 CREATE TRIGGER tr_auditoria_delete_paciente
332 AFTER DELETE ON pacientes
333 FOR EACH ROW
334 BEGIN
335     INSERT INTO log_acciones (
336         usuario, ip_cliente, terminal, rol_activo,
337         accion, tabla, id_afectado, transaccion
338     ) VALUES (
339         CURRENT_USER(),
340         '192.168.0.10',
341         'PC-ADMIN',
342         'admin',
343         'AUDITORÍA: Eliminación de paciente',
344         'pacientes',
345         OLD.id_paciente,
346         CONCAT('DELETE paciente ID=', OLD.id_paciente)
347     );
348 END $$
349 DELIMITER ;
350

```

id_paciente	nombre	fecha_nacimiento	genero	telefono	direccion	correo
4	Ana Martínez	2000-01-30	F	0965432109	Calle Real 101	ana.martinez@mail.com
5	Luis Fernández	1995-06-12	M	0954321098	Av. Las Flores 202	luis.fernandez@mail.com
6	Sofía Torres	1988-09-25	F	0943210987	Calle Luna 303	sofia.torres@mail.com
7	Miguel Ruiz	1975-11-17	M	0932109876	Av. Sol 404	miguel.ruiz@mail.com
8	Laura Castillo	1992-04-08	F	0921098765	Calle Estrella 505	laura.castillo@mail.com

```

351 -- PRUEBA:
352 DELETE FROM pacientes WHERE id_paciente = 8;
353 select * from pacientes;
354

```

id_paciente	nombre	fecha_nacimiento	genero	telefono	direccion	correo
6	Sofía Torres	1988-09-25	F	0943210987	Calle Luna 303	sofia.torres@mail.com
7	Miguel Ruiz	1975-11-17	M	0932109876	Av. Sol 404	miguel.ruiz@mail.com
9	Pedro Morales	1980-08-20	M	0910987654	Av. Mar 606	pedro.morales@mail.com
10	Camila Ríos	1997-02-14	F	0909876543	Calle Río 707	camila.rios@mail.com
* NULL	NULL	NULL	NULL	NULL	NULL	NULL

pacientes 16 x

Figure 14

- ALTER TABLE medicamentos ADD COLUMN stock INT DEFAULT 100;

¿Qué hace?

Modifica la tabla medicamentos para agregar una nueva columna llamada stock, sirve para llevar el control del inventario de medicamentos, es decir, cuántas unidades de cada medicamento hay disponibles.

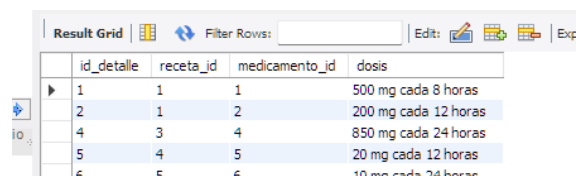
2.tr_control_baja_stock:

¿Qué hace?

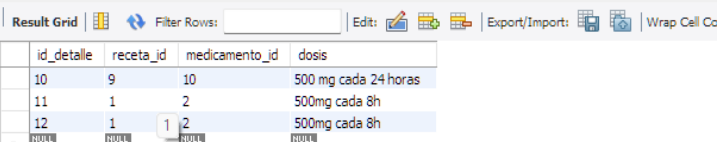
Se ejecuta automáticamente después de insertar un nuevo registro en la tabla detalle_receta. Reduce en una unidad el stock del medicamento correspondiente y deja registro de la operación en log_acciones.

Cómo funciona: Su función es garantizar el control automático del inventario.

```
355 -- 2. Disminución de stock de medicamentos
356 • ALTER TABLE medicamentos ADD COLUMN stock INT DEFAULT 100;
357 DELIMITER $$
358 • CREATE TRIGGER tr_control_baja_stock
359 AFTER INSERT ON detalle_receta
360 FOR EACH ROW
361 BEGIN
362     UPDATE medicamentos
363     SET stock = GREATEST(stock - 1, 0)
364     WHERE id_medimento = NEW.medimento_id;
365
366     INSERT INTO log_acciones (
367         usuario, ip_cliente, terminal, rol_activo,
368         accion, tabla, id_afectado, transaccion
369     ) VALUES (
370         CURRENT_USER(),
371         '192.168.0.10',
372         'PC-ADMIN',
373         'admin',
374         'CONTROL: Baja de stock automática',
375         'detalle_receta',
376         NEW.id_detalle,
377         CONCAT('Medicamento ', NEW.medimento_id, ' -1 stock')
378     );
379 END $$
380 DELIMITER ;
381 • -- PRUEBA:
382 INSERT INTO detalle_receta (receta_id, medicamento_id, dosis)
383 VALUES (1, 2, '500mg cada 8h');
384 • select * from detalle_receta;
385
```



	id_detalle	receta_id	medicamento_id	dosis
▶	1	1	1	500 mg cada 8 horas
	2	1	2	200 mg cada 12 horas
	4	3	4	850 mg cada 24 horas
	5	4	5	20 mg cada 12 horas
	6	5	6	10 mg cada 24 horas



	id_detalle	receta_id	medicamento_id	dosis
	10	9	10	500 mg cada 24 horas
	11	1	2	500mg cada 8h
	12	1	1	500mg cada 8h
•	NULL	NULL	NULL	NULL

Figure 15

3.tr_notificacion_cita:

¿Qué hace?:

Este trigger se activa al insertar una nueva cita médica. Genera un mensaje en la tabla r_notificaciones y deja constancia de la acción en log_acciones, notificando que se ha agendado una nueva consulta médica.

```
5 -- 3. Registrar una cita
6 CREATE TABLE r_notificaciones (
7     id_notificacion INT AUTO_INCREMENT PRIMARY KEY,
8     mensaje TEXT,
9     fecha DATETIME DEFAULT CURRENT_TIMESTAMP
10 );
11 DELIMITER $$
12 CREATE TRIGGER tr_notificacion_cita
13 AFTER INSERT ON citas
14 FOR EACH ROW
15 BEGIN
16     INSERT INTO r_notificaciones (mensaje)
17     VALUES (CONCAT(CHAR(240), CHAR(159), CHAR(147), CHAR(133), ' Nueva cita registrada: paciente ',
18     NEW.paciente_id, ' , médico ', NEW.medico_id, ' , fecha ', NEW.fecha));
19     INSERT INTO log_acciones (
20     usuario, ip_cliente, terminal, rol_activo, accion, tabla, id_afectado, transaccion
21     ) VALUES (
22     CURRENT_USER(),
23     '192.168.0.10',
24     'PC-ADMIN',
25     'admin',
26     'NOTIFICACIÓN: Nueva cita registrada',
27     'citas',
28     NEW.id_cita,
29     CONCAT('INSERT cita ID=', NEW.id_cita)
30     );
31 END $$
32 DELIMITER ;

415 VALUES (1, 1, 1, NOW());
416 select * from citas;
417
```

Result Grid

id_cita	paciente_id	medico_id	tipo_consulta_id	fecha	estado
1	1	1	1	2025-07-01 09:00:00	cancelada
3	3	3	3	2025-07-03 11:00:00	confirmada
4	4	NULL	4	2025-07-04 12:00:00	cancelada
5	5	5	5	2025-07-05 13:00:00	pendiente
6	6	6	6	2025-07-06 14:00:00	confirmada
7	7	7	7	2025-07-07 15:00:00	pendiente

```
413 -- PRUEBA:
414 INSERT INTO citas (paciente_id, medico_id, tipo_consulta_id, fecha)
415 VALUES (1, 2, 1, NOW());
416 select * from citas;
417
```

Result Grid

id_cita	paciente_id	medico_id	tipo_consulta_id	fecha	estado
10	10	10	10	2025-07-10 18:00:00	pendiente
11	1	1	1	2025-08-01 09:00:00	pendiente
12	1	1	1	2025-08-01 19:17:45	pendiente
13	1	1	1	2025-08-01 19:18:50	pendiente
14	1	2	1	2025-08-01 19:19:49	pendiente
NULL	NULL	NULL	NULL	NULL	NULL

citas 20 x

Figure 16

4.tr_historial_diagnostico:

¿Qué hace?

Se activa antes de actualizar un diagnóstico en el historial clínico. Si detecta que el diagnóstico ha cambiado, registra el valor anterior y el nuevo en la tabla historial_cambios, , para garantizar la integridad de la información.

```
DELIMITER $$
• CREATE TRIGGER tr_historial_diagnostico
BEFORE UPDATE ON historial_clinico
FOR EACH ROW
BEGIN
    IF OLD.diagnostico <> NEW.diagnostico THEN
        INSERT INTO historial_cambios (
            id_historial, diagnostico_anterior, diagnostico_nuevo, hash_previo, hash_nuevo
        ) VALUES (
            OLD.id_historial,
            OLD.diagnostico,
            NEW.diagnostico,
            SHA2(OLD.diagnostico, 256),
            SHA2(NEW.diagnostico, 256)
        );

        INSERT INTO log_acciones (
            usuario, ip_cliente, terminal, rol_activo,
            accion, tabla, id_afectado, transaccion
        ) VALUES (
            CURRENT_USER(),
            '192.168.0.10',
            'PC-ADMIN',
            'admin',
            'HISTÓRICO: Cambio de diagnóstico',
            'historial_clinico',
            OLD.id_historial,
            CONCAT('UPDATE diagnóstico historial ID=', OLD.id_historial)
        );
    END IF;
END $$
DELIMITER ;

-- 4. Registro de cambios en diagnóstico
CREATE TABLE historial_cambios (
    id_cambio INT AUTO_INCREMENT PRIMARY KEY,
    id_historial INT,
    diagnostico_anterior TEXT,
    diagnostico_nuevo TEXT,
    hash_previo CHAR(64),
    hash_nuevo CHAR(64),
    fecha_cambio DATETIME DEFAULT CURRENT_TIMESTAMP
);

460 • -- PRUEBA:
461 UPDATE historial_clinico SET diagnostico = 'Nuevo diagnóstico' WHERE id_historial = 1;
462 • select * from historial_clinico
463
464
```

Result Grid						
Filter Rows: <input type="text"/>						
Edit:						
Export/Import:						
Wrap Cell Content:						
id_historial	paciente_id	fecha_registro	diagnostico	tratamiento	observaciones	
1	1	2025-06-01	Nuevo diagnóstico	Medicamentos antihipertensivos	Control mensual	

Figure 17

5.3.3 Consultas de control

- `SELECT * FROM log_acciones ORDER BY fecha DESC;`

¿Qué hace?

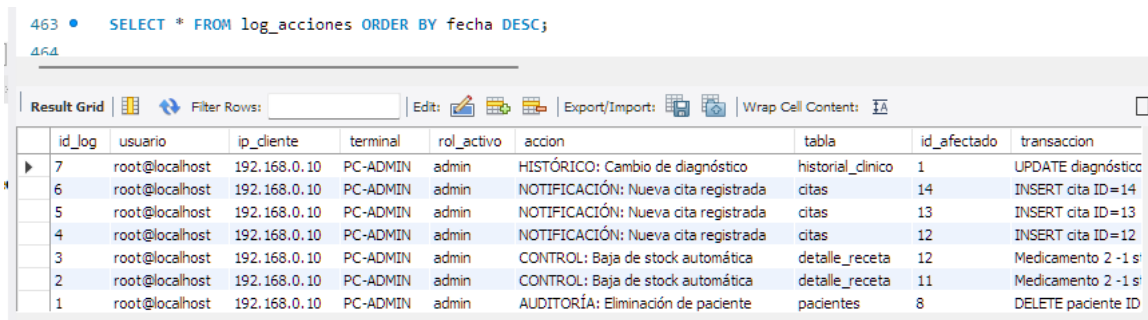
Muestra todos los registros de la tabla `log_acciones`, que contiene el historial de acciones realizadas (INSERT, UPDATE, DELETE, etc.).

Se ordenan de más reciente a más antiguo gracias a `ORDER BY fecha DESC`.

¿Para qué sirve?

Para auditar qué usuarios hicieron qué cambios y cuándo.

Ayuda a llevar el control y trazabilidad de operaciones sensibles en el sistema.



```
463 • SELECT * FROM log_acciones ORDER BY fecha DESC;
```

	id_log	usuario	ip_cliente	terminal	rol_activo	accion	tabla	id_afectado	transaccion
▶	7	root@localhost	192.168.0.10	PC-ADMIN	admin	HISTÓRICO: Cambio de diagnóstico	historial_clinico	1	UPDATE diagnóstico
*	6	root@localhost	192.168.0.10	PC-ADMIN	admin	NOTIFICACIÓN: Nueva cita registrada	citas	14	INSERT cita ID=14
	5	root@localhost	192.168.0.10	PC-ADMIN	admin	NOTIFICACIÓN: Nueva cita registrada	citas	13	INSERT cita ID=13
	4	root@localhost	192.168.0.10	PC-ADMIN	admin	NOTIFICACIÓN: Nueva cita registrada	citas	12	INSERT cita ID=12
	3	root@localhost	192.168.0.10	PC-ADMIN	admin	CONTROL: Baja de stock automática	detalle_receta	12	Medicamento 2 -1 s'
	2	root@localhost	192.168.0.10	PC-ADMIN	admin	CONTROL: Baja de stock automática	detalle_receta	11	Medicamento 2 -1 s'
	1	root@localhost	192.168.0.10	PC-ADMIN	admin	AUDITORÍA: Eliminación de paciente	pacientes	8	DELETE paciente ID

Figure 18

- `SELECT * FROM r_notificaciones ORDER BY fecha DESC;`

¿Qué hace?

Recupera todos los mensajes almacenados en la tabla `r_notificaciones`.

También están ordenados desde los más recientes.

¿Para qué sirve?

Es útil para ver notificaciones generadas automáticamente por triggers, por ejemplo cuando se agenda una nueva cita médica.

```
465 • SELECT * FROM r_notificaciones ORDER BY fecha DESC;
```

```
466
```

	id_notificacion	mensaje	fecha
▶	3	Nueva cita registrada: paciente 1, médico 2, fecha 20...	2025-08-01 19:19:49
	2	Nueva cita registrada: paciente 1, médico 1, fecha 20...	2025-08-01 19:18:50
	1	Nueva cita registrada: paciente 1, médico 1, fecha 20...	2025-08-01 19:17:45
*	NULL	NULL	NULL

Figure 19

- SELECT * FROM historial_cambios ORDER BY fecha_cambio DESC;

¿Qué hace?

Consulta los cambios realizados en los diagnósticos médicos registrados en la tabla historial_clinico.

Se ordenan por fecha_cambio de manera descendente.

¿Para qué sirve?

Permite ver qué diagnóstico fue modificado, cuál era el anterior, cuál es el nuevo y verificar su integridad y control de información clínica sensible.

```
467 • SELECT * FROM historial_cambios ORDER BY fecha_cambio DESC;
```

```
468
```

	id_cambio	id_historial	diagnostico_anterior	diagnostico_nuevo	hash_previo	hash_nuevo	fecha_cambio
▶	1	1	Hipertensión	Nuevo diagnóstico	e62baea0b1ad1d31330b3bb5c34b688037c07c...	e4af405d93f5f46de7b8e0c4ca5eff12a09296...	2025-08-01 19:37:28
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 20

5.4 Índices y Optimización

Se implementaron índices simples y compuestos, análisis con 'EXPLAIN', y simulación de carga con más de 500 registros. Se comparó el rendimiento antes y después de optimizar con mediciones de tiempo de respuesta.

5.4.1 Índices Simples

- Medición de tiempos antes/después de los índices.

```
-- Marcar tiempo inicio
SET @start_time = NOW();
SELECT * FROM citas WHERE paciente_id = 5 ORDER BY fecha DESC;
SELECT TIMEDIFF(NOW(), @start_time) AS tiempo_consulta_citas_antes;
CREATE INDEX idx_pacientes_nombre ON pacientes(nombre);
SET @start_time = NOW();
SELECT * FROM citas WHERE paciente_id = 5 ORDER BY fecha DESC;
SELECT TIMEDIFF(NOW(), @start_time) AS tiempo_consulta_citas_despues;
```

Figure 21

```
SET @start_time = NOW();
```

```
SELECT TIMEDIFF(NOW(), @start_time) AS tiempo_consulta_citas_antes;
```

Usas NOW() para tomar la hora justo antes y después de la consulta.

TIMEDIFF() calcula cuánto tiempo pasó entre esos dos momentos.

Así puedes comparar el rendimiento de la consulta antes y después de crear un índice

- **CREATE INDEX idx_pacientes_nombre ON pacientes(nombre);**

¿Qué hace?

Crea un índice simple sobre la columna nombre de la tabla pacientes, con el nombre idx_pacientes_nombre.

Cómo funciona: Cuando haces una consulta como: `SELECT * FROM pacientes ORDER BY nombre;`

Va directo al dato o rango buscado, sin recorrer todo.

- **CREATE INDEX idx_medicos_especialidad ON medicos(especialidad_id);**

¿Qué hace?

Permite acelerar las consultas que filtran médicos por su especialidad.

Cómo funciona: El índice organiza los datos por especialidad_id, por lo que cuando haces una consulta como:

```
SELECT * FROM medicos WHERE especialidad_id = 3;
```

Usa el índice para ir directamente a las filas que tienen ese valor, sin escanear toda la tabla.

- **CREATE INDEX idx_citas_fecha ON citas(fecha);**

¿Qué hace?

Mejora el rendimiento al **buscar o ordenar citas por fecha**.

Cómo funciona:

```
SELECT * FROM citas ORDER BY fecha DESC;
```

El índice hace que estas operaciones sean mucho más rápidas

- **CREATE INDEX idx_citas_paciente ON citas(paciente_id);**

¿Qué hace?

Crea sobre la columna paciente_id de la tabla citas.

Sirve principalmente para acelerar búsquedas, especialmente en sentencias con JOIN que usan esa columna.

Cómo funciona:

```
SELECT * FROM citas WHERE paciente_id = 5;
```

Busca todas las filas de la tabla citas donde paciente_id sea igual a 5.

5.4.2 Índices compuestos

- **CREATE INDEX idx_citas_paciente_fecha ON citas(paciente_id, fecha);**

¿Qué hace?

Crea un índice que combina las columnas paciente_id y fecha de la tabla citas.

Cómo funciona: Cuando haces consultas que filtran o buscan filas usando ambas columnas o la primera columna paciente_id (por ejemplo, WHERE paciente_id = ? AND fecha >= ?), MySQL puede usar este índice para encontrar rápidamente las filas que coinciden.

Esto es mucho más eficiente que escanear toda la tabla.

```
-- INDICES COMPUESTOS
-- 1. Índice compuesto en citas(paciente_id, fecha)
-- Mejora consultas que buscan citas de un paciente ordenadas o filtradas por fecha.
• CREATE INDEX idx_citas_paciente_fecha ON citas(paciente_id, fecha);
• SET @start_time = NOW();
• SELECT * FROM citas
  WHERE paciente_id = 5 AND fecha >= CURDATE();
• SELECT TIMEDIFF(NOW(), @start_time) AS tiempo_citas_paciente_fecha;
• EXPLAIN SELECT * FROM citas
  WHERE paciente_id = 5 AND fecha >= CURDATE();
```

Figure 22

- **CREATE INDEX idx_historial_paciente_fecha ON historial_clinico(paciente_id, fecha_registro);**

¿Qué hace?

Similar al anterior, este índice combina las columnas paciente_id y fecha_registro de la tabla historial_clinico.

Cómo funciona: Sirve para acelerar consultas que buscan el historial clínico de un paciente en un rango de fechas o para ordenar los registros por fecha.

```
-- 2. Índice compuesto en historial_clinico(paciente_id, fecha_registro)
-- Mejora consultas que buscan el historial clínico de un paciente en orden cronológico.
CREATE INDEX idx_historial_paciente_fecha ON historial_clinico(paciente_id, fecha_registro);

SET @start_time = NOW();
SELECT * FROM historial_clinico
  WHERE paciente_id = 2 AND fecha_registro > '2024-01-01';
SELECT TIMEDIFF(NOW(), @start_time) AS tiempo_historial_paciente_fecha;

EXPLAIN SELECT * FROM historial_clinico
  WHERE paciente_id = 2 AND fecha_registro > '2024-01-01';
```

Figure 23

- **CREATE INDEX idx_detalle_receta ON detalle_receta(receta_id, medicamento_id);**

¿Qué hace?

Índice compuesto sobre las columnas receta_id y medicamento_id de la tabla detalle_receta.

Cómo funciona: Optimiza consultas que buscan detalles de una receta específica y/o un medicamento en particular dentro de esa receta

```

527 -- 3. Índice compuesto en detalle_receta(receta_id, medicamento_id)
528 -- Mejora consultas que buscan medicamentos dentro de una receta específica.
529 -- Tiempo antes de crear el índice
530 • SET @start_time = NOW();
531 • SELECT * FROM detalle_receta
532   WHERE receta_id = 10 AND medicamento_id = 5;
533 • SELECT TIMEDIFF(NOW(), @start_time) AS tiempo_antes;
534 • CREATE INDEX idx_detalle_receta ON detalle_receta(receta_id, medicamento_id);
535 • SET @start_time = NOW();
536 • SELECT * FROM detalle_receta
537   WHERE receta_id = 10 AND medicamento_id = 5;
538 • SELECT TIMEDIFF(NOW(), @start_time) AS tiempo_despues;
539
540 • EXPLAIN SELECT * FROM detalle_receta
541   WHERE receta_id = 10 AND medicamento_id = 5;
542
543

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: ☐

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	detalle_receta	NULL	ref	medicamento_id,idx_detalle_receta	medicamento_id	5	const	1	14,29	Using wher

Figure 24

- **CREATE INDEX idx_log_usuario_fecha ON log_acciones(usuario, fecha);**

¿Qué hace?

Índice en la tabla log_acciones que combina las columnas usuario y fecha.

Cómo funciona: Acelera búsquedas o reportes de acciones filtrando por usuario y ordenando o filtrando por fecha, por ejemplo, para obtener los logs recientes de un usuario.

```

545 -- 4. Índice compuesto en log_acciones(usuario, fecha)
546 -- Mejora consultas que buscan acciones de un usuario ordenadas por fecha.
547 -- Tiempo antes de crear el índice
548 • SET @start_time = NOW();
549 • SELECT * FROM log_acciones
550 WHERE usuario = 'admin'
551 ORDER BY fecha DESC;
552 • SELECT TIMEDIFF(NOW(), @start_time) AS tiempo_antes;
553
554 • CREATE INDEX idx_log_usuario_fecha ON log_acciones(usuario, fecha);
555
556 • SET @start_time = NOW();
557 • SELECT * FROM log_acciones
558 WHERE usuario = 'admin'
559 ORDER BY fecha DESC;
560 • SELECT TIMEDIFF(NOW(), @start_time) AS tiempo_despues;
561
562 • EXPLAIN SELECT * FROM log_acciones
563 WHERE usuario = 'admin'
564 ORDER BY fecha DESC;
565
566

```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	log_acciones	NULL	ref	idx_log_usuario_fecha	idx_log_usuario_fecha	203	const	1	100.00	Backward index scan

EXPLAIN

EXPLAIN se usa para analizar cómo se ejecutará una consulta y verificar si los índices se están utilizando correctamente. Nos ayuda a optimizar el rendimiento y evitar consultas lentas.

EXPLAIN PARA VER RENDIMIENTO

- **EXPLAIN SELECT * FROM citas**
WHERE paciente_id = 5 AND fecha >= CURDATE();

¿Qué hace?

Muestra cómo MySQL busca las citas del paciente 5 a partir de hoy.

Cómo funciona: Si hay un índice en (paciente_id, fecha), MySQL usa ese índice para filtrar rápido sin escanear toda la tabla.

- **EXPLAIN SELECT * FROM historial_clinico**
WHERE paciente_id = 2 AND fecha_registro > '2024-01-01';

¿Qué hace?

Muestra cómo MySQL busca registros del historial clínico del paciente 2 desde cierta fecha.

Cómo funciona: Usa el índice (paciente_id, fecha_registro) para filtrar eficientemente por paciente y fecha.

- **EXPLAIN SELECT * FROM log_acciones**
WHERE usuario = 'admin' ORDER BY fecha DESC;

¿Qué hace?

Muestra cómo MySQL busca acciones del usuario admin y las ordena por fecha descendente.

Cómo funciona: Usa el índice (usuario, fecha) para filtrar y devolver resultados ya ordenados, evitando ordenamiento extra.

5.4.3 Simular carga con 500+ registros

```
DELIMITER $$
CREATE PROCEDURE insertar_pacientes_masivos()
BEGIN
    DECLARE i INT DEFAULT 1;
    WHILE i <= 100 DO
        INSERT INTO pacientes (nombre, fecha_nacimiento, genero)
        VALUES (
            CONCAT('Paciente', i),
            DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND() * 20000) DAY),
            IF(RAND() > 0.5, 'M', 'F')
        );
        SET i = i + 1;
    END WHILE;
END $$
DELIMITER ;
CALL insertar_pacientes_masivos();
SELECT * FROM pacientes ORDER BY id_paciente ;
```

¿Qué hace?

Este procedimiento inserta **100 registros** en la tabla `pacientes`.

Cómo funciona: Empieza un ciclo que se repetirá 100 veces.

- **Insertar 20 médicos.**

```
DELIMITER $$
CREATE PROCEDURE insertar_medicos_masivos()
BEGIN
    DECLARE i INT DEFAULT 1;
    WHILE i <= 20 DO
        INSERT INTO medicos (nombre, cedula, especialidad_id, telefono, correo)
        VALUES (
            CONCAT('Medico', i),
            LPAD(i, 10, '0'),
            FLOOR(1 + RAND()*5),
            CONCAT('09', FLOOR(10000000 + RAND()*89999999)),
            CONCAT('medico', i, '@hospital.com')
        );
        SET i = i + 1;
    END WHILE;
END $$
DELIMITER ;
```

```
CALL insertar_medicos_masivos();  
select * from medicos ;
```

¿Qué hace?

Inserta **20 médicos** con datos generados automáticamente en la tabla médicos.

Cómo funciona: Procedimiento con bucle que inserta registros masivos con datos aleatorios.

- **Insertar 500 citas**

```
DELIMITER $$  
CREATE PROCEDURE insertar_pacientes_masivos()  
BEGIN  
    DECLARE i INT DEFAULT 1;  
    WHILE i <= 100 DO  
        INSERT INTO pacientes (nombre, fecha_nacimiento, genero, telefono, direccion,  
correo)  
        VALUES (  
            CONCAT('Paciente', i),  
            DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND()*20000) DAY),  
            IF(RAND() > 0.5, 'M', 'F'),  
            CONCAT('09', FLOOR(10000000 + RAND()*89999999)),  
            CONCAT('Dirección del paciente ', i),  
            CONCAT('paciente', i, '@correo.com')  
        );  
        SET i = i + 1;  
    END WHILE;  
END $$  
DELIMITER ;  
CALL insertar_pacientes_masivos();  
SELECT * FROM pacientes ORDER BY id_paciente DESC LIMIT 10;
```

¿Qué hace?

Este código crea y ejecuta un procedimiento almacenado llamado insertar_pacientes_masivos que inserta 100 registros nuevos en la tabla pacientes. Cada registro representa un paciente con datos generados

Cómo funciona:

- Se declara una variable i para controlar el ciclo WHILE.
- Mientras i sea menor o igual a 100, se inserta un paciente con datos generados.
- Se incrementa i para continuar el ciclo.
- Termina el procedimiento cuando i supera 100.

- **Insertar 500 registros en historial clínico**

DELIMITER \$\$

CREATE PROCEDURE insertar_historial_clinico()

BEGIN

DECLARE i INT DEFAULT 1;

DECLARE total_pacientes INT;

DECLARE paciente_valido INT;

SELECT COUNT(*) INTO total_pacientes FROM pacientes;

WHILE i <= 500 DO

SELECT id_paciente

INTO paciente_valido

FROM pacientes

ORDER BY RAND()

LIMIT 1;

INSERT INTO historial_clinico (

paciente_id,

fecha_registro,

diagnostico,

tratamiento,

observaciones

) VALUES (

paciente_valido,

DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND()*365) DAY),

CONCAT('Diagnóstico prueba ', i),

CONCAT('Tratamiento prueba ', i),

CONCAT('Observación prueba ', i)

);

SET i = i + 1;

END WHILE;

END \$\$

DELIMITER ;

CALL insertar_historial_clinico();

```
SELECT * FROM historial_clinico ORDER BY id_historial DESC LIMIT 10;
```

¿Qué es?

Es un procedimiento almacenado llamado `insertar_historial_clinico` que inserta 500 registros de prueba en la tabla `historial_clinico`. Cada registro está asociado a un paciente válido de la tabla `pacientes` (elegido al azar).

Cómo funciona: Usa un contador (i) que inicia en 1 y va hasta 500.

En cada ciclo:

- Escoge aleatoriamente un `id_paciente` válido de la tabla `pacientes`.
- Genera datos aleatorios para fecha, diagnóstico, tratamiento y observaciones.
- Inserta esos datos en `historial_clinico`.

Se repite hasta completar 500 inserciones.

5.4.3.1 Anexos

1.

```
566 -- Simular carga con 500+ registros y medir tiempos antes/después de los índices.
567 -- 1. Insertar 100 pacientes
568 DELIMITER $$
569 • CREATE PROCEDURE insertar_pacientes_masivos()
570 BEGIN
571     DECLARE i INT DEFAULT 1;
572     WHILE i <= 100 DO
573         INSERT INTO pacientes (nombre, fecha_nacimiento, genero)
574         VALUES (
575             CONCAT('Paciente', i),
576             DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND() * 20000) DAY),
577             IF(RAND() > 0.5, 'M', 'F')
578         );
579         SET i = i + 1;
580     END WHILE;
581 END $$
582 DELIMITER ;
583 • -- Ejecutar el procedimiento:
584 CALL insertar_pacientes_masivos();
585 • SELECT * FROM pacientes ORDER BY id_paciente ;
```

id_paciente	nombre	fecha_nacimiento	genero	telefono	direccion	correo
9	Pedro Morales	1980-08-20	M	0910987654	Av. Mar 606	pedro.morales@mail.com
10	Camila Ríos	1997-02-14	F	0909876543	Calle Río 707	camila.rios@mail.com
11	Paciente1	1983-06-13	F	NULL	NULL	NULL
12	Paciente2	1982-04-29	M	NULL	NULL	NULL
13	Paciente3	2020-12-10	F	NULL	NULL	NULL
14	Paciente4	1998-03-14	F	NULL	NULL	NULL
15	Paciente5	1985-01-26	F	NULL	NULL	NULL
16	Paciente6	1995-02-14	M	NULL	NULL	NULL

2.

```
589 -- 2. Insertar 20 médicos2. Insertar 20 médicos
590 DELIMITER $$
591 • CREATE PROCEDURE insertar_medicos_masivos()
592 BEGIN
593     DECLARE i INT DEFAULT 1;
594     WHILE i <= 20 DO
595         INSERT INTO medicos (nombre, cedula, especialidad_id, telefono, correo)
596         VALUES (
597             CONCAT('Medico', i),
598             LPAD(i, 10, '0'),
599             FLOOR(1 + RAND()*5),
600             CONCAT('09', FLOOR(10000000 + RAND()*89999999)),
601             CONCAT('medico', i, '@hospital.com')
602         );
603         SET i = i + 1;
604     END WHILE;
605 END $$
606 DELIMITER ;
607 • CALL insertar_medicos_masivos();
608 • select * from medicos ;
```

id_medico	nombre	cedula	especialidad_id	telefono	correo
11	Medico1	0000000001	2	0967377990	medico1@hospital.com
12	Medico2	0000000002	2	0915471877	medico2@hospital.com
13	Medico3	0000000003	1	0947563783	medico3@hospital.com
14	Medico4	0000000004	4	0944465013	medico4@hospital.com
15	Medico5	0000000005	4	0956761227	medico5@hospital.com
16	Medico6	0000000006	2	0948278244	medico6@hospital.com
17	Medico7	0000000007	5	0946435621	medico7@hospital.com
18	Medico8	0000000008	2	0987772640	medico8@hospital.com

3.

```

613 -- Insertar 500 citas
614 DELIMITER $$
615 • CREATE PROCEDURE insertar_pacientes_masivos()
616 BEGIN
617     DECLARE i INT DEFAULT 1;
618     WHILE i <= 100 DO
619         INSERT INTO pacientes (nombre, fecha_nacimiento, genero, telefono, direccion, correo)
620         VALUES (
621             CONCAT('Paciente', i),
622             DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND()*20000) DAY),
623             IF(RAND() > 0.5, 'M', 'F'),
624             CONCAT('09', FLOOR(10000000 + RAND()*89999999)),
625             CONCAT('Dirección del paciente ', i),
626             CONCAT('paciente', i, '@correo.com')
627         );
628         SET i = i + 1;
629     END WHILE;
630 END $$
631 DELIMITER ;
632 • CALL insertar_pacientes_masivos();
633 • SELECT * FROM pacientes ORDER BY id_paciente DESC LIMIT 10;

```

Result Grid						
Filter Rows:						
Edit: Export/Import: Wrap Cell Content: Fetch rows:						
id_paciente	nombre	fecha_nacimiento	genero	telefono	direccion	correo
210	Paciente100	2014-06-16	F	NULL	NULL	NULL
209	Paciente99	1996-09-30	M	NULL	NULL	NULL
208	Paciente98	1984-03-12	F	NULL	NULL	NULL
207	Paciente97	1984-01-01	M	NULL	NULL	NULL
206	Paciente96	1984-11-26	M	NULL	NULL	NULL
205	Paciente95	2022-11-30	F	NULL	NULL	NULL
204	Paciente94	2000-10-18	M	NULL	NULL	NULL
203	Paciente93	2022-03-24	M	NULL	NULL	NULL

pacientes 46 x

Output

4.

```

635 -- Insertar 500 registros en historial clínico
636 DELIMITER $$
637 • CREATE PROCEDURE insertar_historial_clinico()
638 BEGIN
639     DECLARE i INT DEFAULT 1;
640     DECLARE total_pacientes INT;
641     DECLARE paciente_valido INT;
642     SELECT COUNT(*) INTO total_pacientes FROM pacientes;
643     WHILE i <= 500 DO
644         SELECT id_paciente
645         INTO paciente_valido
646         FROM pacientes
647         ORDER BY RAND()
648         LIMIT 1;
649         INSERT INTO historial_clinico (paciente_id,
650             fecha_registro, diagnostico,
651             tratamiento, observaciones
652         ) VALUES (
653             paciente_valido,
654             DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND()*365) DAY),
655             CONCAT('Diagnóstico prueba ', i),
656             CONCAT('Tratamiento prueba ', i),
657             CONCAT('Observación prueba ', i));
658         SET i = i + 1;
659     END WHILE;
660 END $$
661 DELIMITER ;
662 • CALL insertar_historial_clinico();
663 • SELECT * FROM historial_clinico ORDER BY id_historial DESC LIMIT 10;

```

Result Grid						
Filter Rows:						
Edit: Export/Import: Wrap Cell Content: Fetch rows:						
id_historial	paciente_id	fecha_registro	diagnostico	tratamiento	observaciones	
510	41	2025-01-26	Diagnóstico prueba 500	Tratamiento prueba 500	Observación prueba 500	
509	184	2024-12-08	Diagnóstico prueba 499	Tratamiento prueba 499	Observación prueba 499	
508	178	2025-02-03	Diagnóstico prueba 498	Tratamiento prueba 498	Observación prueba 498	
507	191	2024-10-13	Diagnóstico prueba 497	Tratamiento prueba 497	Observación prueba 497	
506	119	2025-03-30	Diagnóstico prueba 496	Tratamiento prueba 496	Observación prueba 496	
505	84	2025-05-27	Diagnóstico prueba 495	Tratamiento prueba 495	Observación prueba 495	
504	72	2025-01-11	Diagnóstico prueba 494	Tratamiento prueba 494	Observación prueba 494	
503	164	2025-01-15	Diagnóstico prueba 493	Tratamiento prueba 493	Observación prueba 493	

historial clinico 47 x

5.5 Seguridad y Roles

Se crearon roles personalizados ('administrador', 'auditor', 'operador', etc.) y usuarios con privilegios restringidos.

5.5.1 Roles

```
-- SEGURIDAD Y ROLES
-- 1. CREACIÓN DE ROLES PERSONALIZADOS
CREATE ROLE administrador;
CREATE ROLE auditor;
CREATE ROLE operador;
CREATE ROLE cliente;
CREATE ROLE proveedor;
CREATE ROLE usuario_final;
```

Figure 25

5.5.2 USUARIOS Y ASIGNACIÓN DE ROLES

```
-- 2. CREACIÓN DE USUARIOS Y ASIGNACIÓN DE ROLES CON NOMBRES PERSONALIZADOS
• CREATE USER 'nicolas_chiguano'@'localhost' IDENTIFIED BY 'Nicolas123';
• CREATE USER 'melany_perugachi'@'localhost' IDENTIFIED BY 'Melany123';
• CREATE USER 'usuario_1'@'localhost' IDENTIFIED BY '1234';
```

Figure 26

```
677 • GRANT administrador TO 'nicolas_chiguano'@'localhost';
678 • GRANT auditor TO 'melany_perugachi'@'localhost';
679 • GRANT operador TO 'usuario_1'@'localhost';
680
```

Figure 27

5.5.3 Rol por defecto

```
-- Activar rol por defecto
• SET DEFAULT ROLE ALL TO 'nicolas_chiguano'@'localhost';
```

Figure 28

5.5.4 Asignación de privilegios

```
-- 3. ASIGNACIÓN DE PRIVILEGIOS CON GRANT
GRANT ALL PRIVILEGES ON hospital.* TO administrador;
GRANT SELECT ON hospital.* TO auditor;
GRANT SELECT, INSERT, UPDATE ON hospital.* TO operador;
```

Figure 29

5.5.5 REVOCACIÓN DE PRIVILEGIOS CON REVOKE

```
-- 4. REVOCACIÓN DE PRIVILEGIOS CON REVOKE
REVOKE INSERT ON hospital.* FROM operador;
```

Figure 30

5.5.5 ENCRIPCIÓN DEMOSTRATIVA

-- Hash con SHA2 y MD5

```
2 -- 5. ENCRIPCIÓN DEMOSTRATIVA
3 -- Hash con SHA2 y MD5
4 • SELECT SHA2('contrasena_segura', 256) AS sha256;
5 • SELECT MD5('contrasena_segura') AS md5;
```

Figure 31

5.5.6 Cifrado y descifrado simétrico con AES

```
697 -- Cifrado y descifrado simétrico con AES
698 • SET @clave = 'mi_clave_secreta';
699 • SET @texto = 'diagnostico confidencial';
700
701 • SET @cifrado = AES_ENCRYPT(@texto, @clave);
702 • SELECT @cifrado;
703 • SELECT AES_DECRYPT(@cifrado, @clave);
704
705
```

```
706 -- 6. EXTRACCIÓN DE DATOS CON REGEXP
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
AES_DECRYPT(@cifrado, @clave)			
BLOB			

Figure 32

5.5.7 VALIDACIÓN DE ENTRADAS CON REGEXP

```
706 -- 6. VALIDACIÓN DE ENTRADAS CON REGEXP
707 -- Solo nombres con letras y espacios
708 • SELECT 'Juan Perez' REGEXP '^[A-Za-z ]+$' AS valido;
709 • SELECT 'Juan123' REGEXP '^[A-Za-z ]+$' AS invalido;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	invalido			
▶	0			

Figure 33

5.5.8 SIMULACIÓN DE INTENTOS FALLIDOS

```
-- 7. SIMULACIÓN DE INTENTOS FALLIDOS
• CREATE TABLE log_intentos (
  id INT AUTO_INCREMENT PRIMARY KEY,
  usuario VARCHAR(50),
  ip_origen VARCHAR(45),
  fecha DATETIME DEFAULT NOW(),
  exito BOOLEAN,
  mensaje TEXT
);
• INSERT INTO log_intentos (usuario, ip_origen, exito, mensaje)
VALUES ('usuario_falso', '192.168.1.100', FALSE, 'Contraseña incorrecta');
```

Figure 34

5.6 Auditoría

La auditoría incluye una tabla 'log_acciones', triggers en operaciones clave (INSERT, UPDATE, DELETE) y trazabilidad de cambios críticos en diagnóstico, incluyendo IP, rol, usuario y acción realizada.

Incluye: log_acciones, triggers de INSERT/UPDATE/DELETE en tablas clave, trazabilidad con hash

Triggers utilizados:

- tr_auditoria_delete_paciente
- tr_control_baja_stock
- tr_notificacion_cita
- tr_historial_diagnostico

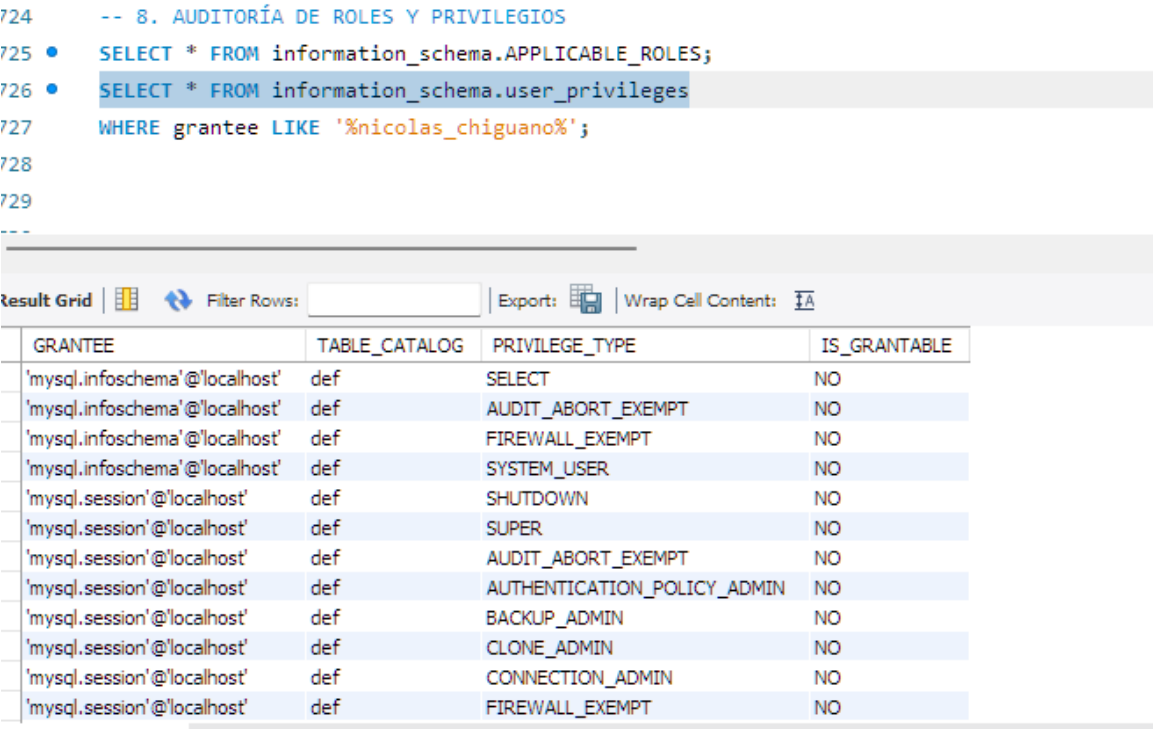


Figure 35

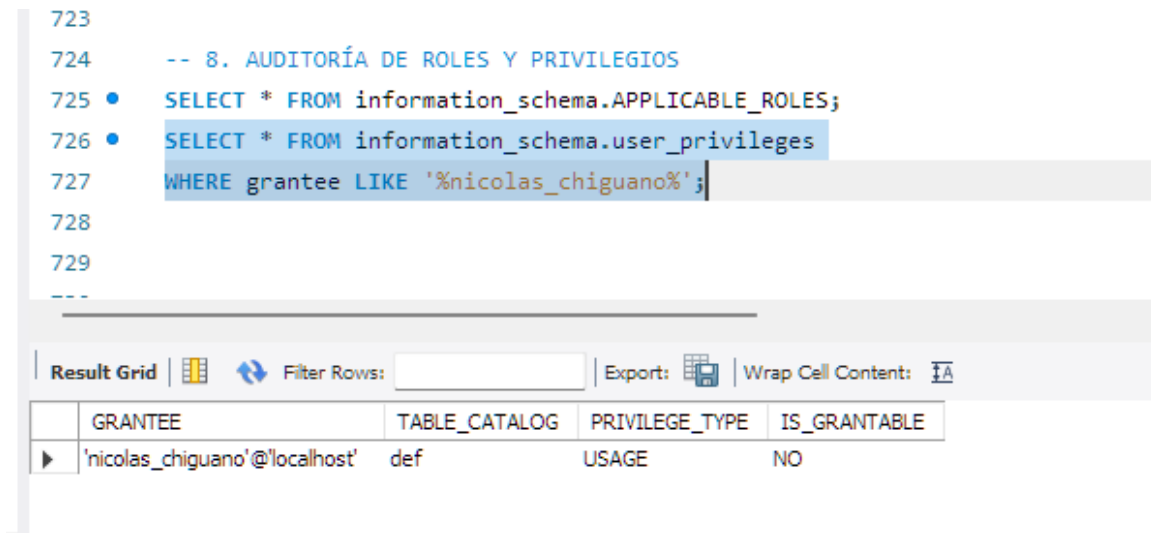


Figure 36

5.7 Respaldo y Recuperación

Se realizaron backups en caliente ('mysqldump') y en frío, además de restauraciones completas desde consola, validando la integridad de la información y asegurando la recuperación ante fallos.

- Backup en caliente (sin detener servidor)

En consola: Desde cmd ingresamos al directorio donde vamos a guardar el backup y luego ejecutamos el siguiente comando e ingresamos la contraseña de mysql.

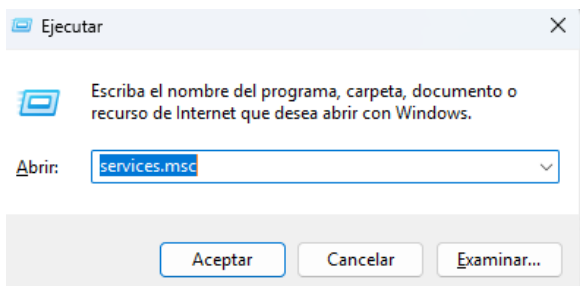
- "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqldump.exe" -u root -p hospital > respaldo_hospital.sql
- Tomar en cuenta que se necesita la ruta de acceso del archivo mysqldump.exe

```

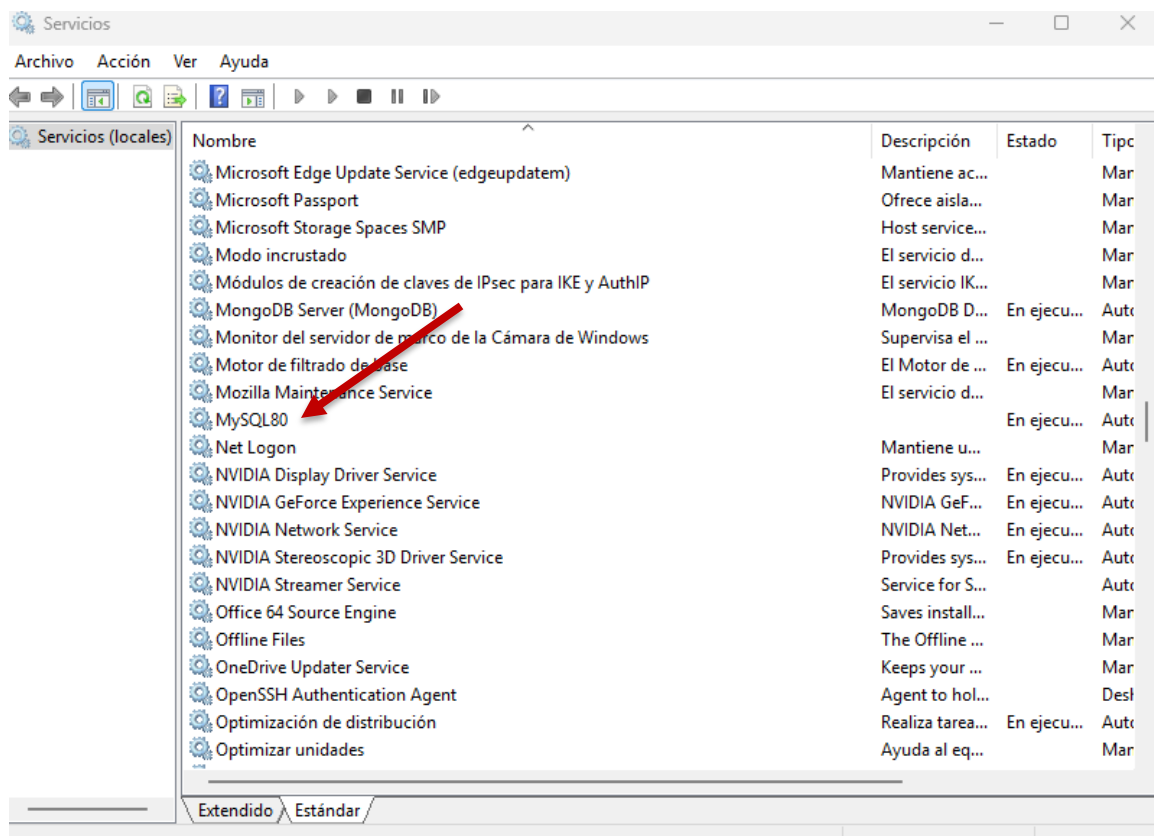
C:\Users\pc\Downloads\ProyectoBDD>"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqldump.exe" -u root -p hospital > resp
aldo_hospital.sql
Enter password: ****
C:\Users\pc\Downloads\ProyectoBDD>|
```

Backup en frío (con servidor detenido):

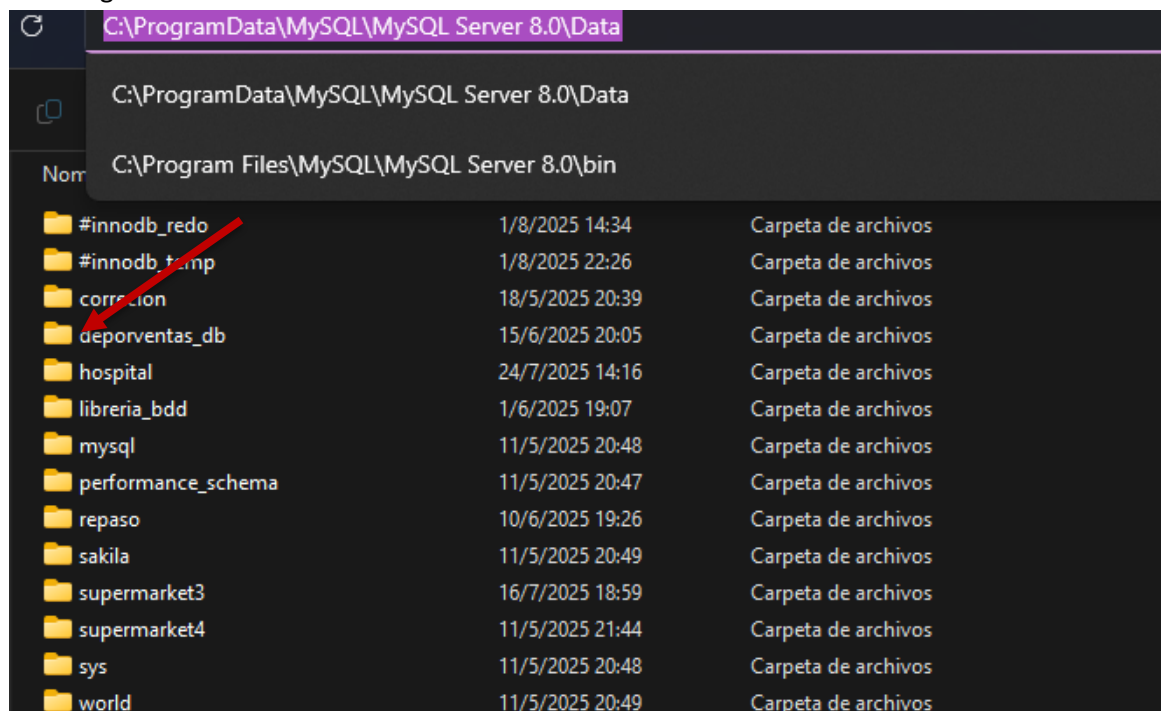
- Detener servicio MySQL ingresando a servicios



Una vez dentro detenemos el servicios MySQL80



- Luego con la ruta por defecto buscamos la carpeta de nuestro proyecto y la pegamos en otros lugar.



- Por ultimo volvemos a reiniciar el servicio

Restauración desde consola:

- Para ello usaremos la ruta del backup realizado junto con el comando `mysql -u root -p hospital < respaldo_hospital.sql`
- Ingresamos la contraseña de Mysql y la restauración esta hecha.

```
C:\Users\pc\Downloads\ProyectoBDD>"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql.exe" -u root -p hospital < respaldo_hospital.sql
Enter password: ****
C:\Users\pc\Downloads\ProyectoBDD>
```

3.- SEGURIDAD ANTE SQL INJECTION

Simulación:

- `SELECT * FROM usuarios WHERE usuario = " OR '1'='1'; -- vulnerable`

Prevención (procedimiento con parámetros):

DELIMITER \$\$

CREATE PROCEDURE buscar_usuario_seguro(IN user_input VARCHAR(50))

BEGIN

SELECT * FROM usuarios WHERE usuario = user_input;

END \$\$

DELIMITER;

Validación previa:

DELIMITER \$\$

CREATE FUNCTION validar_usuario(u VARCHAR(50)) RETURNS BOOLEAN

BEGIN

RETURN u REGEXP '^[a-zA-Z0-9_.-]+\$';

END \$\$

DELIMITER ;

Uso seguro:

- `CALL buscar_usuario_seguro('juan');`

4.- MONITOREO Y RENDIMIENTO

Tamaño de tablas e índices:

SHOW TABLE STATUS FROM hospital;

SHOW INDEX FROM pacientes;

Crecimiento de registros semanal:

SELECT WEEK(fecha), COUNT(*) FROM log_acciones GROUP BY WEEK(fecha);

Consultas lentas

Activar log:

- SET GLOBAL slow_query_log = 'ON';
- SET GLOBAL long_query_time = 2;

Consultar:

- SELECT * FROM mysql.slow_log ORDER BY start_time DESC;

Registro de uso de funciones/procedimientos:

Puede auditarse manualmente desde log_acciones si se inserta trazabilidad (ya implementado en triggers).

5.- PROTECCIÓN DE DATOS Y GESTIÓN CRÍTICA

Cifrado de correos:

- ALTER TABLE pacientes ADD COLUMN correo_cifrado VARBINARY(255);
- UPDATE pacientes SET correo_cifrado = AES_ENCRYPT(correo, 'clave_secreta');
- SELECT AES_DECRYPT(correo_cifrado, 'clave_secreta') FROM pacientes;

Anonimización:

- UPDATE pacientes SET nombre = CONCAT('PAC-', id_paciente);

Integridad lógica:

DELIMITER \$\$

CREATE FUNCTION validar_dosis(dosis TEXT) RETURNS BOOLEAN

BEGIN

RETURN dosis REGEXP '^[0-9]+mg cada [0-9]+h\$';

END \$\$

DELIMITER ;

6.- SIMULACIÓN DE PERFILES PROFESIONALES

Administrador BD

Verifica índices:

SHOW INDEX FROM medicamentos;

Programar tarea automática (MySQL Event):

DELIMITER \$\$

CREATE EVENT ev_backup_diario

ON SCHEDULE EVERY 1 DAY STARTS CURRENT_TIMESTAMP

DO

BEGIN

INSERT INTO log_acciones (usuario, ip_cliente, terminal, rol_activo, accion, tabla, id_afectado, transaccion)

VALUES ('event_scheduler', '127.0.0.1', 'SISTEMA', 'admin', 'BACKUP automático diario', 'N/A', NULL, 'Evento ejecutado');

END \$\$

DELIMITER ;

Arquitecto BD

Revisión de integridad:

- SHOW TABLES;
- SHOW CREATE TABLE historial_clinico;

Oficial de Seguridad

Creación de roles (simulado):

- CREATE USER 'auditor'@'localhost' IDENTIFIED BY 'segura123';
- GRANT SELECT ON hospital.* TO 'auditor'@'localhost';

Revisión de logs:

- SELECT * FROM log_acciones ORDER BY fecha DESC;

Desarrollador de Consultas:

- CREATE VIEW vw_resumen_citas AS
- SELECT medico_id, COUNT(*) AS total_citas FROM citas GROUP BY medico_id;

Analista de Datos

KPIs:

- SELECT COUNT(*) AS total_pacientes FROM pacientes;
- SELECT COUNT(*) AS citas_mes_actual FROM citas WHERE MONTH(fecha) = MONTH(NOW());

Usuario Final

Acceso a vistas controladas:

- SELECT * FROM vw_resumen_citas;

7.- VALIDACIÓN FINAL DEL PROYECTO

Checklist (manual):

- ✓ Triggers implementados
- ✓ Bitácora activa
- ✓ Procedimientos seguros
- ✓ Cifrado aplicado

6. Resultados y Pruebas

Se verificó la ejecución correcta de procedimientos, funciones, triggers y reportes. Las medidas de seguridad protegieron contra inyecciones SQL y accesos no autorizados. Las consultas optimizadas mejoraron significativamente su rendimiento.

7. Conclusiones

El proyecto permitió aplicar conceptos aprendidos de bases de datos en un entorno realista, cumpliendo con buenas prácticas de diseño, seguridad y rendimiento. La integración de procedimientos, funciones, triggers y control de roles garantiza la escalabilidad y confiabilidad del sistema.