



ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

BASE DE DATOS



ASIGNATURA:

Base de Datos

PROFESOR:

Ing. Yadira Franco

PERÍODO ACADÉMICO:

2025 – A

PROYECTO FINAL BASE DE DATOS

TÍTULO:

- Gestión de Base de datos de un Hospital

ESTUDIANTES

Ramos De La Cruz Wilmer Adrian
Escobar Obando Edison Gabriel

FECHA DE REALIZACIÓN:

27 / Julio / 2025

FECHA DE ENTREGA:

3 / Agosto / 2025

CALIFICACIÓN OBTENIDA:

FIRMA DEL PROFESOR:

Nombre General del Proyecto: Base de datos Hospital en PostgreSQL

Sistema Integral de Gestión para Entornos Profesionales con Bases de Datos Relacionales

Objetivo General:

Diseñar e implementar un sistema profesional de base de datos, utilizando MySQL, PostgreSQL o SQL Server (asignado por grupo), que abarque desde el modelado lógico hasta la protección contra ataques, aplicando buenas prácticas de rendimiento, seguridad, respaldo, programación avanzada y roles profesionales reales en una temática asignada.

Link de GitHub:

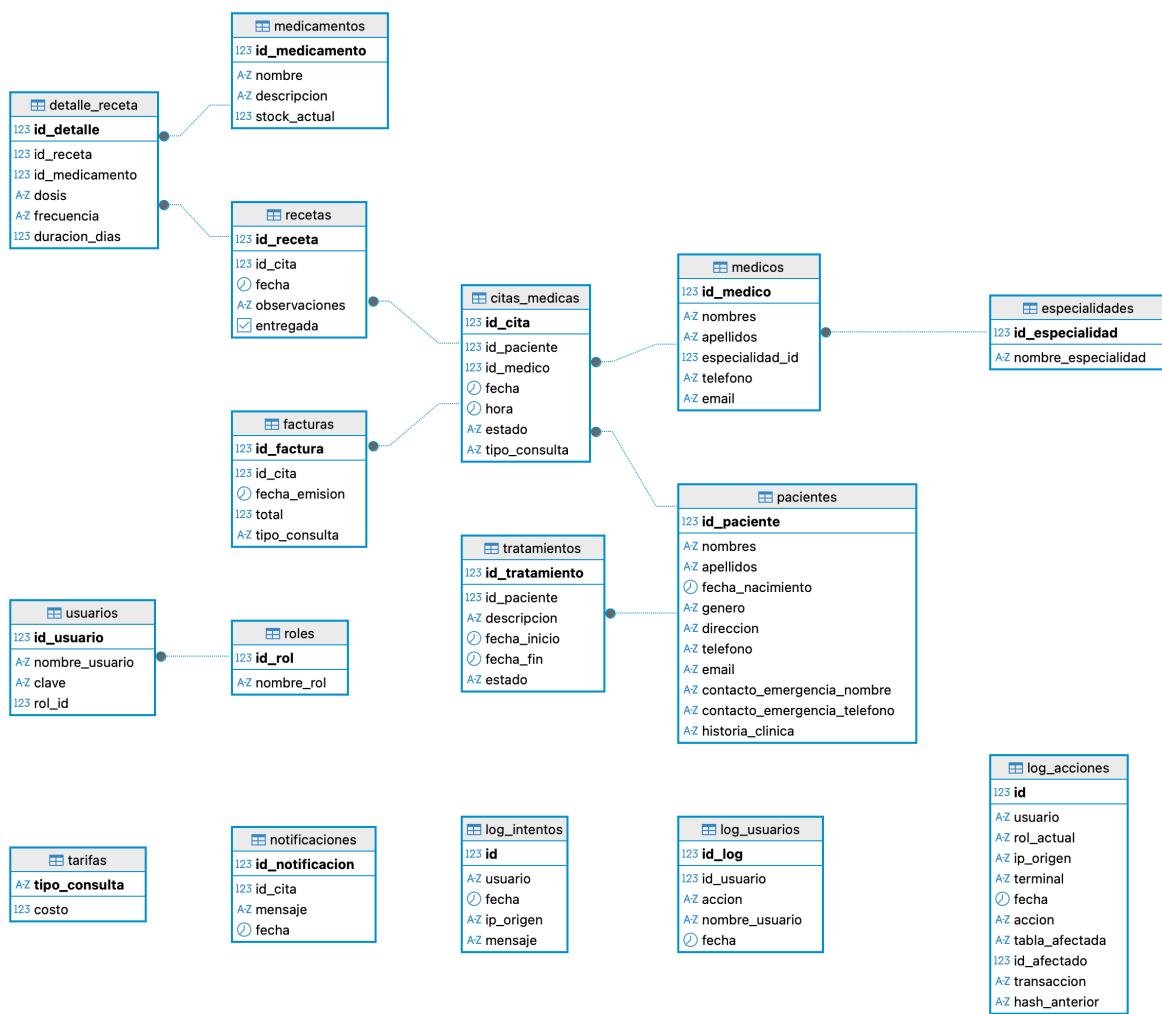
https://github.com/WilmerRamos21/Proyecto_Base_de_Datos_Hospital.git

Requerimientos Técnicos Específicos

Modelado y Normalización

Normalización: Las tablas de este proyecto se encuentran en la Tercera Forma Normal ya que pasaron por el filtro de la Primera y Segunda Forma Normal dando como resultado tablas que no contiene más de un dato dentro de una columna o datos repetidos que provoquen confusión al momento de la creación de los parámetros de cada tabla y que cada tabla se relacione de forma correcta mediante sus claves primarias y sus claves foráneas.

Modelado: Esta es el diagrama que tiene la base y las relaciones correspondientes para cada tabla:



Como se puede visualizar dentro de la imagen las relaciones están correctas y sus claves relacionan las tablas correspondientes para facilitar el manejo de los registros y consulta de todas las tablas.

Integridad y Restricciones

Es el modelado para las tablas se aplicó correctamente las restricciones como:

- SERIAL: Para que aumente la clave primaria cuando se agregue un nuevo registro.
- CHECK: Para los catálogos o ENUM para MySQL.
- NOT NULL: Para evitar el ingreso de datos nulos en los registros.
- UNIQUE: Para los correos electrónicos, nombres de usuario, especialidades.
- DEFAULT: Para que los campos tengan un valor por defecto como por ejemplo el estado de una cita en ‘Pendiente’.

No aplicamos el ON DELETE CASCADE, ON UPDATE CASCADE, SET NULL en este proyecto para mantener la integridad de la base ya que si por algún error se da una eliminación de un registro este puede desencadenar una reacción en cadena que puede dañar la integridad de la base dando como resultado datos obsoletos dentro de la base sin previo aviso, por ello se optó el uso de CASCADE.

Definición de Estructura Base

La cantidad de tablas funcionales de la base de datos inicial fue de 12 tablas, pero a medida que se avanzaba en el proyecto se agregaron tablas adicionales dando como resultado final un total de 16 tablas, las tablas adicionales hacen referencia a el lugar donde se guardan los triggers como auditorias o intentos de conexión para la simulación de SSL dentro de PostgreSQL.

> postgres localhost:5432	
>	ssl_test localhost:5433
>	└ Databases
>	ssl_test
>	└ Schemas
>	hospital
>	└ Tables
>	citas_medicas 232K
>	detalle_receta 40K
>	especialidades 40K
>	facturas 24K
>	log_acciones 16K
>	log_intentos 32K
>	log_usuarios 8K
>	medicamentos 32K
>	medicos 24K
>	notificaciones 72K
>	pacientes 64K
>	recetas 48K
>	roles 24K
>	tarifas 24K
>	tratamientos 64K
>	usuarios 40K

Procedimientos Almacenados

Inserción con validación cruzada

- 1) Insertar una cita médica validando que el paciente y el médico existen, de ser el caso que uno de los dos no existe lanza un mensaje de error con “Paciente no existe” o “Medico no existe”, si ambos existen se registra la cita médica.

Casos de error:

The screenshot shows two separate SQL result panes. The top pane shows an attempt to insert a cita with patient ID 1 and doctor ID 200, resulting in an error: "SQL Error [P0001]: ERROR: Medico no existe." The bottom pane shows an attempt to insert a cita with patient ID 100 and doctor ID 2, resulting in an error: "SQL Error [P0001]: ERROR: Paciente no existe." Both errors point to line 16 at RAISE.

Caso de ingreso válido:

```
CALL insertar_cita (1, 2, '2025-08-01', '12:00', 'Pendiente', 'General');
```

506	506	1	2	2025-08-01	12:00:00	Pendiente	General
-----	-----	---	---	------------	----------	-----------	---------

Generación de reportes por periodo

- 2) Mostrar citas de los médicos por un rango de fechas determinado facilitando la búsqueda y control de las citas médicas realizadas.

The screenshot shows the output of a stored procedure call. The procedure is named "reporte_citas_medicas" and takes parameters for doctor ID (1), start date ('2025-01-01'), and end date ('2025-07-20'). The output lists numerous medical appointments with details like patient ID, date, and time.

```
CALL reporte_citas_medicas(1, '2025-01-01', '2025-07-20');

Cita: 24, Paciente: 2, Fecha: 2025-01-03, Hora: 14:45:07
Cita: 44, Paciente: 2, Fecha: 2025-06-14, Hora: 13:18:12
Cita: 45, Paciente: 1, Fecha: 2025-02-07, Hora: 08:07:00
Cita: 47, Paciente: 3, Fecha: 2025-03-22, Hora: 15:07:37
Cita: 52, Paciente: 2, Fecha: 2025-05-04, Hora: 13:27:12
Cita: 62, Paciente: 3, Fecha: 2025-01-13, Hora: 12:23:13
Cita: 67, Paciente: 5, Fecha: 2025-05-01, Hora: 11:10:22
Cita: 76, Paciente: 1, Fecha: 2025-06-07, Hora: 08:26:30
Cita: 78, Paciente: 4, Fecha: 2025-01-06, Hora: 08:50:28
Cita: 80, Paciente: 1, Fecha: 2025-06-16, Hora: 12:48:53
Cita: 81, Paciente: 4, Fecha: 2025-04-28, Hora: 10:05:16
Cita: 95, Paciente: 3, Fecha: 2025-06-23, Hora: 11:01:30
Cita: 106, Paciente: 2, Fecha: 2025-02-27, Hora: 09:47:13
Cita: 116, Paciente: 1, Fecha: 2025-05-13, Hora: 08:49:52
Cita: 117, Paciente: 5, Fecha: 2025-03-15, Hora: 09:24:43
Cita: 118, Paciente: 5, Fecha: 2025-03-07, Hora: 13:12:59
Cita: 121, Paciente: 1, Fecha: 2025-02-10, Hora: 14:06:01
Cita: 126, Paciente: 4, Fecha: 2025-02-07, Hora: 08:25:25
Cita: 151, Paciente: 5, Fecha: 2025-07-06, Hora: 14:47:27
Cita: 155, Paciente: 4, Fecha: 2025-01-12, Hora: 13:55:30
Cita: 163, Paciente: 3, Fecha: 2025-06-15, Hora: 12:05:33
Cita: 181, Paciente: 4, Fecha: 2025-01-22, Hora: 08:13:24
```

Actualizaciones masivas por condición

- 3) Actualizar estado de tratamiento cuya fecha de fin ya pasó.

Inicialmente se encuentra así:

Grid	123 ↗ id_tratamiento	123 ↗ id_paciente	AZ descripción	① fecha_inicio	① fecha_fin	AZ estado
1	1	1	Tratamiento respiratorio para asma	2025-07-01	2025-07-31	Activo
2	2	3	Control de presión arterial	2025-06-01	2025-07-15	Finalizado
3	3	2	Terapia para migraña crónica	2025-07-10	2025-08-10	Activo

Al aplicar el procedimiento almacenado los tratamientos que cuya fecha ya pasó debe quedar como finalizado:

Grid	123 ↗ id_tratamiento	123 ↗ id_paciente	AZ descripción	① fecha_inicio	① fecha_fin	AZ estado
1	2	3	Control de presión arterial	2025-06-01	2025-07-15	Finalizado
2	3	2	Terapia para migraña crónica	2025-07-10	2025-08-10	Activo
3	1	1	Tratamiento respiratorio para asma	2025-07-01	2025-07-31	Finalizado

Eliminación segura

- 4) Elimina a un paciente que no tiene ninguna cita médica registrada, en caso de que exista una cita, se lanza un mensaje de error impidiendo que se borre un cliente con citas médicas registradas en la base de datos.

En este caso se intentó eliminar el paciente con ID 4 pero no se llegó a completar su eliminación debido a que tiene citas registradas:

-- Llamada de ejemplo
CALL eliminar_paciente_seguro(4)

Facturación automática (inserción con validación + transacción)

- 5) Generar factura automáticamente desde una cita si no tiene una factura asociada, en este caso se verifica en primer lugar que existe la cita y luego si existe su respectiva factura, si no existe la factura se procede a obtener los datos relevantes de la cita para elaborar una facturación automática:

En este caso se procede con un ejemplo:

-- Llamada de ejemplo
CALL generar_facturaAutomatica(1);

Y con una consulta a la tabla facturas se muestra la factura creada:

Grid	123 ↗ id_factura	123 ↗ id_cita	① fecha_emision	123 total	AZ tipo_consulta
1	1	3	2025-07-18	20	General
2	2	5	2025-07-19	50	Emergencia
3	3	1	2025-08-01	20	General

Funciones Definidas por el Usuario

1. Función para calcular la edad del paciente mediante su ID y que el valor de la edad se retorne como entero:

```
-- Probar el funcionamiento de la función
SELECT calcular_edad(1) AS Edad_Paciente;
```

Results 1

	edad_paciente
1	30

2. Función para calcular la duración de días de un tratamiento mediante el ID de tratamiento retornando la cantidad de días como entero entre un rango de fechas de inicio y fin:

```
-- Ejemplo de uso:
SELECT duracion_tratamiento(2);
```

Results 1

	duracion_tratamiento
1	44

3. Función para verificar si un paciente tiene tratamiento activo o no, mediante el ID del paciente:

```
-- Ejemplo de uso:
SELECT estado_tratamiento_paciente(2);
```

Results 1

	estado_tratamiento_paciente
1	Activo

Triggers

1. Auditoría con tabla de log para usuarios, para mantener un control sobre los registros insertados, eliminados o modificados, combinando su funcionalidad con una función que determina cuando el trigger debe ser ejecutado:
En este caso se lo puso a prueba con un INSERT a usuarios y este es el resultado del trigger que guarda las acciones ejecutadas y se lo puede consultar revisando la tabla de log_usuarios:

```
-- Ejemplo de uso del trigger
INSERT INTO usuarios(nombre_usuario, clave, rol_id)
VALUES ('receptionista_0', 'rcp123', 1);
SELECT * FROM log_usuarios;
```

usuarios 1
:T * FROM log_usuarios Enter a SQL expression to filter results (use Ctrl+Space)
123 id_log 123 id_usuario AZ accion A-Z nombre_usuario DI fecha

id_log	id_usuario	accion	nombre_usuario	fecha
1	8	INSERT	receptionista_0	2025-08-02 09:11:04.626

2. Trigger de control automático de stock de los medicamentos en el inventario al registrar un detalle_receta.

El estado de la tabla medicamentos antes de insertar el medicamento solicitado y entregado al paciente:

Grid	123 ↗ id_medicamento	AZ nombre	AZ descripcion	123 stock_actual
1	1	Paracetamol 500mg	Analgésico y antipirético	200
2	2	Amoxicilina 500mg	Antibiótico de amplio espectro	100
3	3	Ibuprofeno 400mg	Analgésico, antiinflamatorio	150
4	4	Omeprazol 20mg	Protector gástrico	80
5	5	Loratadina 10mg	Antihistamínico	120

El estado de la tabla medicamentos cuando se inserto un registro a la tabla detalle_receta con el medicamento indicado:

The screenshot shows the Oracle SQL Developer interface. At the top, there is a code editor window containing two SQL statements:

```

-- Ejemplo de uso del trigger
INSERT INTO recetas (id_cita, fecha, observaciones, entregada) VALUES
(6, '2025-07-10', 'Tratamiento para el dolor de cabeza.', TRUE);
-- INSERT INTO detalle_receta (id_receta, id_medicamento, dosis, frecuencia, duracion_dias) VALUES
(3, 1, '1 tableta', 'Cada 8 horas', 4);

```

Below the code editor is a results grid titled "medicamentos 1 X". It displays the following data:

Grid	123 ↗ id_medicamento	AZ nombre	AZ descripcion	123 stock_actual
1	2	Amoxicilina 500mg	Antibiótico de amplio espectro	100
2	3	Ibuprofeno 400mg	Analgesico, antiinflamatorio	150
3	4	Omeprazol 20mg	Protector gástrico	80
4	5	Loratadina 10mg	Antihistamínico	120
5	1	Paracetamol 500mg	Analgesico y antipirético	199

Se puede ver que el medicamento “Paracetamol 500mg” se redujo en 1 de su stock.

3. Trigger de notificación simulada: El objetivo de este trigger es simular la creación de una notificación para una cita médica de estado “Pendiente” que se guardara en la tabla de notificaciones:

The screenshot shows the Oracle SQL Developer interface. At the top, there is a code editor window containing two SQL statements:

```

-- Ejemplo de funcionamiento del trigger
INSERT INTO citas_medicas (id_paciente, id_medico, fecha, hora, estado, tipo_consulta) VALUES
(2, 2, '2025-01-03', '13:00:40', 'Pendiente', 'General');
SELECT * FROM notificaciones;

```

Below the code editor is a results grid titled "notificaciones 1 X". It displays the following data:

Grid	123 ↗ id_notificacion	123 id_cita	AZ mensaje	fecha
1	1		9 Se ha agendado una nueva cita para el paciente Mateo Cuevas	2025-08-01 12:23:29.293
2	2		10 Se ha agendado una nueva cita para el paciente Esteban Rojas	2025-08-01 12:23:29.293
3	3		17 Se ha agendado una nueva cita para el paciente Esteban Rojas	2025-08-01 12:23:29.293
4	4		18 Se ha agendado una nueva cita para el paciente Esteban Rojas	2025-08-01 12:23:29.293
5	5		19 Se ha agendado una nueva cita para el paciente Fernanda León	2025-08-01 12:23:29.293
6	6		20 Se ha agendado una nueva cita para el paciente Fernanda León	2025-08-01 12:23:29.293

Índices y Optimización

- Crear índices simples para campos clave en WHERE, JOIN, ORDER BY.
- Crear índices compuestos para combinaciones comunes

```
-- Índices para mejorar búsquedas por filtros
CREATE INDEX idx_paciente_genero ON pacientes(genero);
CREATE INDEX idx_citas_estado ON citas_medicas(estado);
CREATE INDEX idx_citas_fecha ON citas_medicas(fecha);
CREATE INDEX idx_tratamientos_estado ON tratamientos(estado);

-- Índices en claves foráneas (optimizan JOINs)
CREATE INDEX idx_citas_paciente ON citas_medicas(id_paciente);
CREATE INDEX idx_citas_medico ON citas_medicas(id_medico);
CREATE INDEX idx_recetas_cita ON recetas(id_cita);
CREATE INDEX idx_detalle_medicamento ON detalle_receta(id_medicamento);

-- 2. ÍNDICES COMPUESTOS
-- Búsqueda de citas por médico y rango de fecha
CREATE INDEX idx_citas_medico_fecha ON citas_medicas(id_medico, fecha);

-- Búsqueda de tratamientos por paciente y estado
CREATE INDEX idx_tratamientos_paciente_estado ON tratamientos(id_paciente, estado);

-- Búsqueda de citas por tipo y estado (uso frecuente en reportes)
CREATE INDEX idx_citas_tipo_estado ON citas_medicas(tipo_consulta, estado);
```

Se han creado varios índices a lo largo del proyecto que afectan únicamente a columnas clave o que se mencionan mucho a lo largo de la gestión de los registros.

- Ahora vamos a monitorear el rendimiento de una consulta de sin índice y una con índice:

```
-- Antes de aplicar índices
EXPLAIN ANALYZE
SELECT * FROM citas_medicas
WHERE id_medico = 2 AND fecha BETWEEN '2025-07-01' AND '2025-07-31';
```

Grid	AZ QUERY PLAN
1	Bitmap Heap Scan on citas_medicas (cost=4.82..10.76 rows=11 width=43) (actual time=0.058..0.098 rows=8 loops=1)
2	Recheck Cond: ((fecha >= '2025-07-01'::date) AND (fecha <= '2025-07-31'::date))
3	Filter: (id_medico = 2)
4	Rows Removed by Filter: 46
5	Heap Blocks: exact=5
6	-> Bitmap Index Scan on idx_citas_fecha (cost=0.00..4.81 rows=54 width=0) (actual time=0.027..0.027 rows=54 loop:1)
7	Index Cond: ((fecha >= '2025-07-01'::date) AND (fecha <= '2025-07-31'::date))
8	Planning Time: 2.153 ms
9	Execution Time: 0.267 ms

Como se puede observar se tiene un tiempo de planeamiento de 2.153 ms y tiempo de ejecución de 0.267 ms sin usar índices.

Ahora se prueba el tiempo de la consulta con el índice creado:

Results 1 X	
Grid	EXPLAIN ANALYZE SELECT * FROM c ↕ Enter a SQL expression to filter results (use Ctrl+Space)
Text	AZ QUERY PLAN
1	Bitmap Heap Scan on citas_medicas (cost=4.41..9.61 rows=11 width=43) (actual time=0.101..0.114 rows=8 loops=1)
2	Recheck Cond: ((id_medico = 2) AND (fecha >= '2025-07-01'::date) AND (fecha <= '2025-07-31'::date))
3	Heap Blocks: exact=5
4	-> Bitmap Index Scan on idx_citas_medico_fecha (cost=0.00..4.41 rows=11 width=0) (actual time=0.090..0.090 rows=8 loops=1)
5	Index Cond: ((id_medico = 2) AND (fecha >= '2025-07-01'::date) AND (fecha <= '2025-07-31'::date))
6	Planning Time: 0.532 ms
7	Execution Time: 0.173 ms

Se observa una mejora en el tiempo de planeamiento con un total de 0.532 ms y un tiempo de ejecución con 0.173 ms que es mucho menor al de la misma consulta pero sin indices.

- Se realizo una simulación de carga de 500 registros en la tabla de citas_medicas para ver el tiempo de resultado de la ejecución de los índices creados.

Results 1 X	
Grid	EXPLAIN ANALYZE SELECT * FROM c ↕ Enter a SQL expression to filter results (use Ctrl+Space)
Text	AZ QUERY PLAN
1	Bitmap Heap Scan on citas_medicas (cost=4.78..10.71 rows=62 width=43) (actual time=0.141..0.174 rows=70 loops=1)
2	Recheck Cond: (((tipo_consulta)::text = 'General'::text) AND ((estado)::text = 'Pendiente'::text))
3	Heap Blocks: exact=5
4	-> Bitmap Index Scan on idx_citas_tipo_estado (cost=0.00..4.77 rows=62 width=0) (actual time=0.125..0.126 rows=70 loops=1)
5	Index Cond: (((tipo_consulta)::text = 'General'::text) AND ((estado)::text = 'Pendiente'::text))
6	Planning Time: 0.249 ms
7	Execution Time: 0.207 ms

Se puede ver que el tiempo de búsqueda pese a tener 500 registros es elevado, dando por conclusión que los índices mejoran el tiempo que se tarda en realizar una consulta, pero se debe tener cuidado de solo crear índices solo a columnas de uso frecuente.

Seguridad y Roles

Gestión de privilegios a los roles creados.

Creación de roles personalizados como:

- Administrador con todos los privilegios.
- Auditor solo con permisos de lectura.
- Operador permitir realizar consulta, inserción y actualización.
- Cliente solo de lectura.
- Proveedor solo inserta y actualiza.
- Usuario final solo de acceso básico.

```

-- 1. Creación de Roles Personalizados
-- Rol administrador (superusuario)
CREATE ROLE administrador LOGIN PASSWORD 'admin123' SUPERUSER;

-- Rol auditor
CREATE ROLE auditor LOGIN PASSWORD 'auditor123';

-- Rol operador
CREATE ROLE operador LOGIN PASSWORD 'operador123';

-- Rol cliente (solo lectura)
CREATE ROLE cliente LOGIN PASSWORD 'cliente123';

-- Rol proveedor (inserta y actualiza)
CREATE ROLE proveedor LOGIN PASSWORD 'proveedor123';

-- Rol usuario_final (solo acceso básico)
CREATE ROLE usuario_final LOGIN PASSWORD 'usuario123';

-- 2. Gestión de Privilegios con GRANT / REVOKE
-- Dar privilegios de solo lectura al auditor
GRANT SELECT ON ALL TABLES IN SCHEMA public TO auditor;

-- Permitir al operador hacer SELECT, INSERT y UPDATE
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA public TO operador;

-- Permitir al cliente solo ver datos
GRANT SELECT ON pacientes, tratamientos TO cliente;

-- Revocar UPDATE al cliente por seguridad
REVOKE UPDATE ON tratamientos FROM cliente;

```

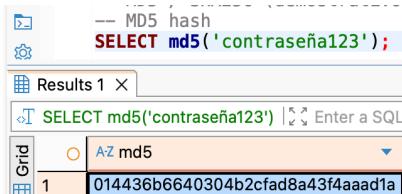
Verificar los roles creados:

-- Consultar los roles creados					
	AZ rolname	rolsuper	rolcreaterole	rolcreatedb	rolcanlogin
10	pg_execute_server_p	[]	[]	[]	[]
11	pg_signal_backend	[]	[]	[]	[]
12	pg_checkpoint	[]	[]	[]	[]
13	pg_maintain	[]	[]	[]	[]
14	pg_use_reserved_co	[]	[]	[]	[]
15	pg_create_subscripti	[]	[]	[]	[]
16	postgres	[v]	[v]	[v]	[v]
17	administrador	[v]	[]	[]	[v]
18	auditor	[]	[]	[]	[v]
19	operador	[]	[]	[]	[v]
20	cliente	[]	[]	[]	[v]
21	proveedor	[]	[]	[]	[v]
22	usuario_final	[]	[]	[]	[v]
23	repcionista	[]	[]	[]	[v]
24	medico	[]	[]	[]	[v]

Se puede ver que los roles fueron creados de acuerdo a los requerimientos.

Encriptación con SHA2 y MD5 de forma demostrativa y uso de pgcrypto.

- En primer lugar se crea la extensión de pgcrypto para el entorno de PostgreSQL.
CREATE EXTENSION IF NOT EXISTS pgcrypto;
- Luego la encriptación demostrativa con MD5:



```
-- MD5 hash
SELECT md5('contraseña123');
```

Results 1 ×

SELECT md5('contraseña123') | Enter a SQL expression to filter results (use Ctrl+Space)

Grid	AZ md5
1	014436b6640304b2cfad8a43f4aaad1a

- Despues la encriptación demostrativa con SHA256:



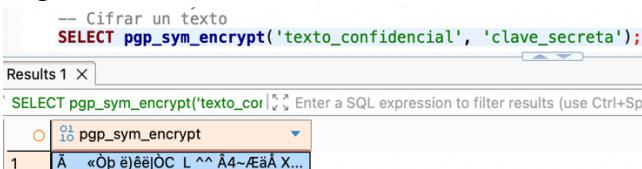
```
-- SHA256
SELECT encode(digest('contraseña123', 'sha256'), 'hex');
```

Results 1 ×

SELECT encode(digest('contraseña123', 'sha256'), 'hex') | Enter a SQL expression to filter results (use Ctrl+Space)

Grid	AZ encode
1	8e7ab8d9fe3b324acdd1f76735eea350ea61ac24cbd17e5446946e5a4c71d999

- Seguido del cifrado del texto:



```
-- Cifrar un texto
SELECT pgp_sym_encrypt('texto_confidencial', 'clave_secreta');
```

Results 1 ×

SELECT pgp_sym_encrypt('texto_confidencial', 'clave_secreta') | Enter a SQL expression to filter results (use Ctrl+Space)

Grid	AZ pgp_sym_encrypt
1	Ã“Ob ejÃ©lOC L ^Â4-ÆääÃ...X...

- Y por ultimo el descifrado demostrativo del texto



```
-- Descifrar el texto
SELECT pgp_sym_decrypt(pgp_sym_encrypt('texto_confidencial', 'clave_secreta'), 'clave_secreta');
```

Results 1 ×

SELECT pgp_sym_decrypt(pgp_sym_encrypt('texto_confidencial', 'clave_secreta'), 'clave_secreta') | Enter a SQL expression to filter results (use Ctrl+Space)

Grid	AZ pgp_sym_decrypt
1	texto_confidencial

Simulación de conexión SSL para asegurar el tráfico entre cliente y servidor

1. Para simular SSL en DBeaver se debe dirigir primero al archivo de postgresql.conf pero como en mi sistema operativo Mac no se encuentran directamente estos servicios descargue Docker para usar PostgreSQL de forma sencilla, para ello cree desde la terminal la carpeta postgres_ssl que guardara la configuración necesaria para simular SSL.
2. Dentro de esta carpeta se encuentran los archivos necesarios como server.key, postgresql.conf entre otros:

```
Last login: Sat Aug  2 10:15:11 on ttys000
(base) adrian_ramos@MacBook-Air-de-Adrian ~ % cd postgres_ssl
(base) adrian_ramos@MacBook-Air-de-Adrian postgres_ssl % ls
certs           docker-compose.yml    ssl
config          postgresql.conf
(base) adrian_ramos@MacBook-Air-de-Adrian postgres_ssl %
```

3. Se deben crear y editar los archivos postgresql.conf y en mi caso docker-compose.yml y deben contener los siguiente:

postgresql.conf:

```
listen_addresses = '*'
port = 5432
ssl = on
ssl_cert_file = '/var/lib/postgresql/certs/server.crt'
ssl_key_file = '/var/lib/postgresql/certs/server.key'

shared_preload_libraries = 'pg_stat_statements'
```

docker-compose.yml:

```
services:
  postgres:
    image: postgres:17
    container_name: postgres_ssl
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: ssl_test
    ports:
      - "5433:5432"
    volumes:
      - ./postgresql.conf:/etc/postgresql/postgresql.conf
      - ./config/pg_hba.conf:/etc/postgresql/pg_hba.conf
      - ./ssl:/var/lib/postgresql/certs
    command: ["postgres", "-c", "config_file=/etc/postgresql/postgresql.conf"]
```

Terminal Bottom:

```
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Pg  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is   ^V Next Pg  ^U UnCut Text ^T To Spell
```

4. Se deben asignar las rutas correctas de las credenciales necesarias y luego iniciar el servicio de SSL, como se puede observar el servicio de SSL esta corriendo correctamente:

Docker Desktop interface showing the Containers list. The sidebar includes options like Ask Gordon, Containers (selected), Images, Volumes, Builds, Models, MCP Toolkit, Docker Hub, Docker Scout, and Extensions. The main area shows the following containers:

Name	Container ID	Image	Port(s)	CPU (%)	Last seen	Actions
pgsqlpreview	bb999a5c89f9	postgres	5432:5432	0%	1 day ago	▶ ⋮ trash
sqlpreview	c4994ed191d3	mssql/serv	1433:1433	0%	1 month ago	▶ ⋮ trash
postgres_ssl	-	-	-	0.01%	2 hours ago	▶ ⋮ trash
postgres_ssl_83e620912fa3	83e620912fa3	postgres:1	5433:5432	0.01%	2 hours ago	▶ ⋮ trash

5. En DBeaver se debe habilitar la opcion de SSL y seleccionar “required” y se puede observar el servicio activo de la siguiente forma mediante una consulta:

Connection "ssl_test" configuration

Connection settings

PostgreSQL connection settings

Main | Advanced | Driver properties | **SSL** X

All SSL parameters are optional.
You must specify SSL certificates if they are required by your server configuration.
Settings on this page override Driver properties.

DBeaver does not verify SSL configuration and relies on the driver implementation.
Please refer to the driver documentation for more information.

Parameters

CA Certificate:

Client Certificate:

Client Private Key:

Advanced

SSL mode: **require**

SSL Factory:

Results 1 ×

```

    SELECT ssl FROM pg_stat_ssl WHERE pid = pg_backend_pid();
    SHOW ssl;
  
```

Results 1 ×

SHOW ssl | ↵ ↴ Enter a SQL expression to filter results (use Ctrl+Space)

ssl	AZ ssl
on	on

6. Se simula los intentos fallidos o sospechosos para ello se crea una tabla de log_intentos y un registro para simular el intento desde una IP desconocida para guardar dichos intentos:

Luego, simula un trigger para registrar el intento:

```

  INSERT INTO log_intentos (usuario, ip_origen, mensaje)
  VALUES ('usuario_falso', '192.168.0.105', 'Intento fallido de acceso');
  
```

SELECT * FROM log_intentos;

log_intentos 1 ×

SELECT * FROM log_intentos | ↵ ↴ Enter a SQL expression to filter results (use Ctrl+Space)

id	usuario	fecha	ip_origen	mensaje
1	usuario_falso	2025-08-01 12:24:32.871	192.168.0.105	Intento fallido de acceso

Validaciones de entrada usando expresiones regulares para restringir caracteres peligrosos o no válidos.

- Para ello se crea una función que evalúe los caracteres permitidos para los correos electrónicos de los usuarios, ejemplo de uso.
- Correo valido:

```
-- Uso:  
SELECT validaremail('lucia.herrera@gmail.com'); -- OK  
results 1 X  
SELECT validaremail('lucia.herrera@c'); Enter a SQL expression to filter results (
```

	<input checked="" type="checkbox"/> validaremail
	[v]

- Correo incorrecto:

```
SELECT validaremail('correo#incorrecto'); -- ERROR  
results 1 X  
SELECT validaremail('correo#incorecc'); Enter a SQL expression to filter results  
SQL Error [P0001]: ERROR: Correo inválido  
Where: PL/pgSQL function validaremail(character varying) line 6 at RAISE  
Error position:
```

Revisión de roles y privilegios

Mediante una consulta verificamos los roles creados con sus privilegios

```
④-- Consultar los roles creados  
SELECT rolname, rolsuper, rolcreaterole, rolcreatedb, rolcanlogin  
FROM pg_roles;
```

	rolname	rolsuper	rolcreaterole	rolcreatedb	rolcanlogin
10	pg_execute_server_p	[]	[]	[]	[]
11	pg_signal_backend	[]	[]	[]	[]
12	pg_checkpoint	[]	[]	[]	[]
13	pg_maintain	[]	[]	[]	[]
14	pg_use_reserved_co	[]	[]	[]	[]
15	pg_create_subscripti	[]	[]	[]	[]
16	postgres	[v]	[v]	[v]	[v]
17	administrador	[v]	[]	[]	[v]
18	auditor	[]	[]	[]	[v]
19	operador	[]	[]	[]	[v]
20	cliente	[]	[]	[]	[v]
21	proveedor	[]	[]	[]	[v]
22	usuario_final	[]	[]	[]	[v]
23	repcionista	[]	[]	[]	[v]
24	medico	[]	[]	[]	[v]

Ahora para verificar los privilegios activos lo hacemos desde esta consulta:

```
④ -- Privilegios activos
SELECT grantee, privilege_type, table_name
FROM information_schema.role_table_grants
WHERE grantee IN ('auditor', 'operador', 'cliente');

role_table_grants 1 ×
SELECT grantee, privilege_type, table_ Enter a SQL expression to filter results
④ AZ grantee ▾ AZ privilege_type ▾ AZ table_name ▾
1 cliente SELECT pacientes
2 auditor SELECT pacientes
3 cliente SELECT tratamientos
```

Auditoría

Para la auditoría se creó una tabla log_acciones que actuará como una tabla centralizada que captará y guardará los registros borrados, modificados o insertados, mediante una función asociada a los triggers de tablas clave reportando de esa manera el usuario, su dirección IP, la terminal, fecha de la acción y la tabla afectada.

A continuación, se muestra una consulta a la tabla centralizada:

```
④ -- Verificar los procesos ejecutados consultando la tabla centralizada log_acciones
SELECT * FROM log_acciones;

log_acciones 1 ×
SELECT * FROM log_acciones Enter a SQL expression to filter results (use Ctrl+Space)
④ 123 ↗ id ▾ AZ usuario ▾ AZ rol_actual ▾ AZ ip_origen ▾ AZ terminal ▾ ⏴ fecha ▾ AZ accion ▾ AZ tabla_ 
1 1 postgres postgres 192.168.65.1/32 localhost 2025-08-01 14:42:19.784 UPDATE tratamien
2 2 postgres postgres 192.168.65.1/32 localhost 2025-08-01 14:52:59.156 INSERT facturas
```

Seguridad ante SQL Injection

Simulación de formulario vulnerable con el cual el intruso puede acceder fácilmente a los registros sensibles de la base de datos:

```
④ -- 1. Simulación de SQL Injection
-- Simula un ataque como este:
SELECT * FROM usuarios WHERE nombre_usuario = 'admin' AND clave = '' OR '1'='1';

usuarios 1 ×
SELECT * FROM usuarios WHERE nor Enter a SQL expression to filter results (use Ctrl+Space)
④ 123 ↗ id_usuario ▾ AZ nombre_usuario ▾ AZ clave ▾ 123 ↗ rol_id ▾
1 1 recepcion1 123456 1
2 2 doctor_cvera medico123 2
3 3 farmacia1 farm@2025 3
4 4 auditor_admin audit2025 4
5 5 doctor_aparedes neurologia123 2
6 6 admin_general admin1234 4
7 8 recepcionista_0 rcp123 1
```

Simulación con pseudocódigo desde un archivo.ipynb a la base de datos de postgres:

Mediante Python se puede crear una conexión simulada a la base de datos PostgreSQL. Donde el intruso probara ingresando un input para el nombre del usuario como es el caso más obvio utilizara admin y tratara de acceder desde ahí para obtener los datos sensibles de la tabla.

Y una vez que los obtenga solo realizara un print de los datos a los que accedió.

Prevención

Para prevenir dichos ataques se puede implementar validación de expresiones regulares para restringir caracteres especiales.

Vistas que solo expongan las columnas necesarias.

Procedimientos almacenados con parámetros de validados internamente.

– Ejemplo de validación:

```
CREATE OR REPLACE FUNCTION validar_usuario(p_usuario TEXT)
RETURNS BOOLEAN AS $$

BEGIN
    IF p_usuario ~ '^[a-zA-Z0-9]+$' THEN
        RETURN TRUE;
    ELSE
        RAISE EXCEPTION 'Usuario contiene caracteres inválidos.';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

```
SQL Error [P0001]: ERROR: Usuario contiene caracteres
inválidos.
```

```
Where: PL/pgSQL function validar_usuario(text) line 6
at RAISE
```

Error position:

Monitoreo y Rendimiento

Consulta de tamaño de las tablas, uso de disco:

```
-- Tamaño total de la base de datos
SELECT pg_size_pretty(pg_database_size('ssl_test'));
```

Results 1 ×

```
SELECT pg_size_pretty(pg_database_size()); Enter a SQL expression to filter result:
```

1	8859 kB
1	8859 kB

```
--- Tamaño de cada tabla
SELECT relname AS tabla, pg_size_pretty(pg_total_relation_size(relid)) AS tamaño
FROM pg_catalog.pg_statio_user_tables
ORDER BY pg_total_relation_size(relid) DESC;
```

pg_statio_user_tables 1 ×

```
SELECT relname AS tabla, pg_size_pretty(pg_total_relation_size(relid)) AS tamaño; Enter a SQL expression to filter results (use Ctrl+Space)
```

1	AZ tabla	AZ tamaño
1	citas_medicinas	224 kB
2	notificaciones	72 kB
3	pacientes	64 kB
4	tratamientos	64 kB
5	recetas	48 kB
6	usuarios	40 kB
7	especialidades	40 kB
8	detalle_receta	40 kB
9	log_acciones	32 kB
10	medicamentos	32 kB
11	log_intentos	32 kB
12	roles	24 kB
13	medicos	24 kB
14	tarifas	24 kB
15	facturas	24 kB

Para el control de registros por semana modificamos la tabla de pacientes y agregamos la columna de fecha_creacion con valor por defecto NOW(), luego realizamos una consulta para ver la cantidad de registros por semana:

```
ALTER TABLE pacientes
ADD COLUMN fecha_creacion TIMESTAMP DEFAULT now();

SELECT date_trunc('week', fecha_creacion) AS semana, COUNT(*) AS registros
FROM pacientes
GROUP BY semana
ORDER BY semana DESC;
```

Results 1 ×			
SELECT date_trunc('week', fecha_creacion) AS semana, COUNT(*) AS registros FROM pacientes GROUP BY semana ORDER BY semana DESC;			
○	semana	123 registros	▼
1	2025-07-28 00:00:00.000	5	

Evaluación de consultas lentas mediante esta búsqueda usando EXPLAIN ANALYZE:

--- 3. Evaluar consultas lentas:	
<code>EXPLAIN ANALYZE SELECT FROM pacientes WHERE apellidos = 'Herrera';</code>	
Results 1 ×	
EXPLAIN ANALYZE SELECT FROM pa	Enter a SQL expression to filter results (use Ctrl+Space)
○	AZ QUERY PLAN
1	Seq Scan on pacientes (cost=0.00..1.06 rows=1 width=0) (actual time=0.058..0.059 rows=1 loops=1)
2	Filter: ((apellidos)::text = 'Herrera'::text)
3	Rows Removed by Filter: 4
4	Planning Time: 0.624 ms
5	Execution Time: 0.195 ms

Registro del uso de procedimientos de funciones, procedimientos, recursos

- Revisamos el catálogo de pg_stat_user_functions pero no existe ninguno:

```
-- Revisa el catálogo pg_stat_user_functions:
SELECT funcname, calls, total_time
FROM pg_stat_user_functions
ORDER BY total_time DESC;
```

pg_stat_user_functions 1 ×			
SELECT funcname, calls, total_time F			
○	AZ funcname	123 calls	▼
		123 total_time	▼

- Luego creamos la extensión de pg_stat_statements para mostrar el registro de las funciones y su número de usos:

```
-- Usa pg_stat_statements para monitorear uso general:
CREATE EXTENSION IF NOT EXISTS pg_stat_statements;
```

```
-- Visualizar el archivo de configuracion
SHOW config_file;
```

```
-- Mostrar las librerias
SHOW shared_preload_libraries;
```

```
-- Mirar las pg_stat_statements de monitoreo
SELECT query, calls, total_exec_time
FROM pg_stat_statements
```

```
ORDER BY total_exec_time DESC
LIMIT 5;
```

- Y como se puede observar las pg_stat_statements que se utilizan:

Grid	AZ query	123 calls	123 total_exec_time
1	SELECT c.relname,a.*pg_catalog.pg_get_expr(ad.adbin, ad.adrelid, \$2) as def_value,dsc.description,dep.obj	55	37.54674
2	SELECT t.oid,t.*c.relkind,format_type(nullif(t.typtype,\$1),t.typtypmod) as base_type_name, d.descripti	4	17.993999
3	SELECT c.oid,c.*d.description,pg_catalog.pg_get_expr(c.relpartbound, c.oid) as partition_expr, pg_catalog.p	12	16.059041
4	SELECT * FROM citas_medicas	13	9.675877
5	select c.oid,pg_catalog.pg_total_relation_size(c.oid) as total_rel_size,pg_catalog.pg_relation_size(c.oid) as r	2	8.573626

Protección de Datos y Gestión Crítica

Demostración de cifrado de datos sensibles como emails, contraseñas, entre otras:

```
--- Ideal para contraseñas (no se descifran, solo se comparan).
--- Almacenamiento (usando SHA-256)
INSERT INTO usuarios (nombre_usuario, clave, rol_id)
VALUES ('repcion_1', digest('clave123', 'sha256'),1);
```

```
--- Verificación
SELECT * FROM usuarios
WHERE nombre_usuario = 'repcion_1';
```

Grid	123 id_usuario	AZ nombre_usuario	AZ clave	123 rol_id
1	9	repcion_1	\x5ac0852e770506dcd80f1a36d20ba7878bf82244b836d9324593bd14bc56dc5	1

Se muestra la clave encriptada y almacenada en la tabla de usuarios para proteger datos sensibles.

Demostración de cifrado del correo electrónico con AES:

Grid	AZ nombres	AZ apellidos	AZ fecha_nacimiento	AZ genero	AZ direccion	AZ telefono	AZ email
1	Lucía	Herrera	1995-04-10	F	Calle A #123	0981122334	lucia.herrera@gmail.com
2	Mateo	Cueva	2001-01-25	M	Av. Central 45	0999988776	mateo.cueva@gmail.com
3	Esteban	Rojas	1980-11-15	M	Calle Sur 12	0977744112	esteban.rojas@gmail.com
4	Valentina	Silva	2005-07-02	F	Barrio Norte 77	0955544332	valen.silva@gmail.com
5	Fernanda	León	1990-09-10	F	Av. del Sol 88	0967890123	fernanda.leon@gmail.com
6	Juan	Herrera	1995-04-10	M	Calle A #123	0981122334	\xc30d040703022e2

Demostración de descifrado del correo:

```
-- Cifrar manualmente y descifrar en la misma consulta
SELECT nombres, apellidos, pgp_sym_decrypt(pgp_sym_encrypt('juan.herrera@mail.com'::text, 'clave_segura'::text),
FROM pacientes
WHERE id_paciente = 6;|
```

Grid	AZ nombres	AZ apellidos	AZ email
1	Juan	Herrera	juan.herrera@mail.com

Simulación de cifrado anónimo de datos sensibles (irreversible)

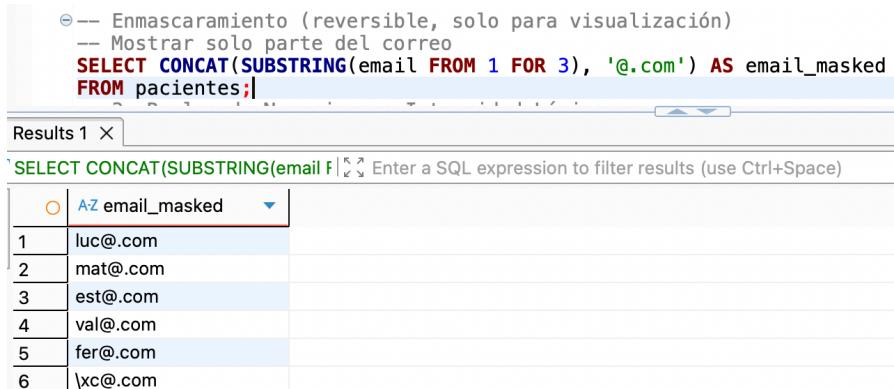
Como se trata de una acción irreversible solo se explicará que realiza un reemplazo total de los nombres y la eliminación de los emails, solo es una parte de explicación por lo que no se realizó dicho proceso para mantener la integridad de la base.

```
-- Anonimización (irreversible)
-- Reemplazo total
UPDATE pacientes
SET nombres = 'Paciente_' || id;

-- Eliminación de campos sensibles
UPDATE pacientes
SET email = NULL, direccion = NULL;
```

Simulación de enmascaramiento (reversible)

En este caso se procede a enmascarar los correos para su protección:



The screenshot shows a SQL query being run in a database interface. The query is:

```
SELECT CONCAT(SUBSTRING(email FROM 1 FOR 3), '@.com') AS email_masked
FROM pacientes;
```

The results table has two columns: 'email' and 'email_masked'. The 'email' column contains the original email addresses, and the 'email_masked' column contains the masked versions where the first three characters of each email are followed by '@.com'. The results are:

	email	email_masked
1	luc@.com	luc@.com
2	mat@.com	mat@.com
3	est@.com	est@.com
4	val@.com	val@.com
5	fer@.com	fer@.com
6	\xc@.com	\xc@.com

Implementación de integridad lógica

- Para lograr una implementación de integridad lógica se debe usar funciones y procedimientos almacenados que lo sostengan.
- No se puede cubrir solo con restricciones Check o Foreign Keys.
- Se puede crear una función que evita crear citas duplicadas con el mismo médico el mismo día.
- Una función que evite el ingreso de menores de edad según el contexto y que solo permita a personas mayores de edad.
- Otra función que valide los correos electrónicos de los usuarios.
- Validación de los nombres y que no permita caracteres especiales.

Aplicando dichas validaciones y restricciones se puede lograr una integridad alta dentro de la base de datos y sus registros.

Simulación de Perfiles Profesionales en la base de datos

1. Administrador de BD

Tiene como función principal las siguientes operaciones:

- Mantenimiento de los índices.

-- Ver tamaño y uso de índices								
SELECT * FROM pg_stat_user_indexes;								
t_user_indexes 1 < T * FROM pg_stat_user_indexe Enter a SQL expression to filter results (use Ctrl+Space)								
123 relid	123 indexrelid	AZ schemaname	AZ relname	AZ indexrelname	123 idx_scan	123 last_idx_scan		
16,391	16,396	hospital	roles	roles_pkey	8	2025-08-02 12:56:00.76		
16,399	16,403	hospital	usuarios	usuarios_pkey	0			
16,399	16,405	hospital	usuarios	usuarios_nombre_usuar	1	2025-08-02 12:56:29.8		
16,413	16,417	hospital	especialidades	especialidades_pkey	5	2025-08-01 12:20:19.2		
16,413	16,419	hospital	especialidades	especialidades_nombre_	0			
16,422	16,426	hospital	medicos	medicos_pkey	508	2025-08-01 14:27:27.8:		
16,434	16,441	hospital	pacientes	pacientes_pkey	8	2025-08-01 12:20:19.2		
16,444	16,450	hospital	citas_medicas	citas_medicas_pkey	8	2025-08-02 09:21:19.6		
16,463	16,470	hospital	medicamentos	medicamentos_pkey	5	2025-08-02 09:22:39.16		
16,473	16,480	hospital	recetas	recetas_pkey	3	2025-08-01 12:20:19.2		
16,488	16,493	hospital	detalle_receta	detalle_receta_pkey	0			
16,505	16,510	hospital	tarifas	tarifas_pkey	1	2025-08-01 14:52:59.16		

- Respaldos y restauraciones.
- Tareas programadas.
- Monitoreo del sistema.
- Ver las estadísticas de uso de disco

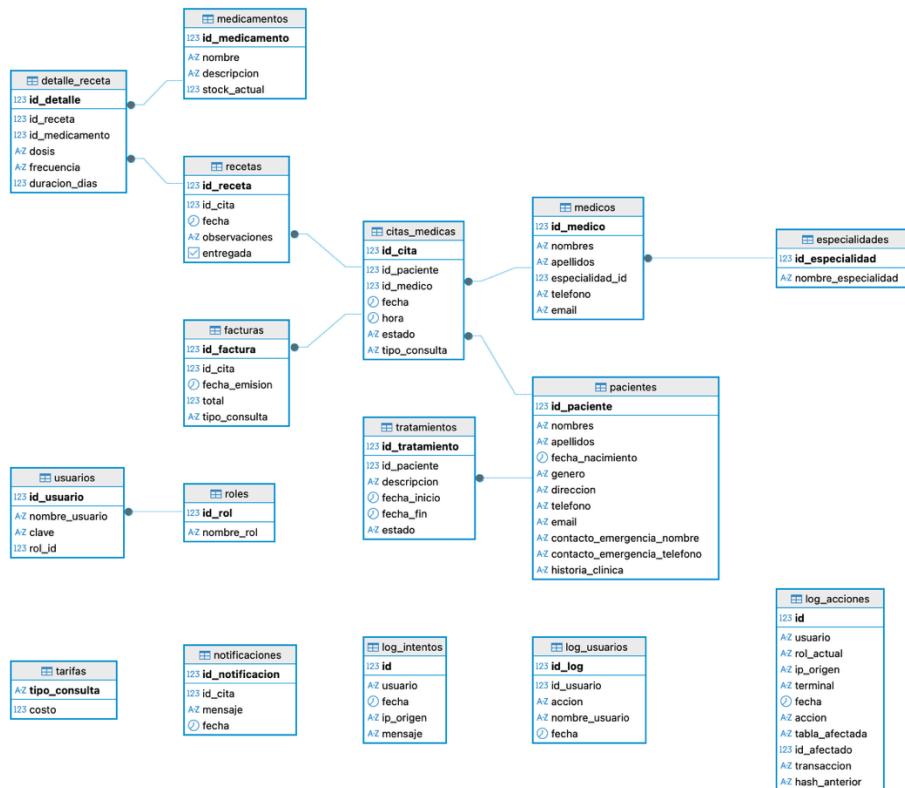
AZ tabla	AZ tamaño
citas_medicas	224 kB
notificaciones	72 kB
tratamientos	64 kB
pacientes	64 kB
recetas	48 kB
usuarios	40 kB
especialidades	40 kB
detalle_receta	40 kB
medicamentos	32 kB
log_acciones	32 kB
log_intentos	32 kB
roles	24 kB

2. Arquitecto de BD

Tiene como funciones principales:

- Diseño lógico y físico (MER y MR)
- Definición de los nombres, tipos y claves.
- Modularidad y escalabilidad.
- Verificación de integridad.

Diagrama de relación de las tablas:



--- Verificación de integridad

--- Verificar claves foráneas

```

SELECT conname, conrelid::regclass AS tabla, conrefid::regclass AS referencia
FROM pg_constraint
WHERE contype = 'f';
    
```

constraint 1 X

:T conname, conrelid::regclass / Enter a SQL expression to filter results (use Ctrl+Space)

AZ conname	AZ tabla	AZ referencia
usuarios_rol_id_fkey	usuarios	roles
medicos_especialidad	medicos	especialidades
citas_medicas_id_pac	citas_medicas	pacientes
citas_medicas_id_me	citas_medicas	medicos
recetas_id_cita_fkey	recetas	citas_medicas
detalle_receta_id_rec	detalle_receta	recetas
detalle_receta_id_me	detalle_receta	medicamentos
facturas_id_cita_fkey	facturas	citas_medicas
tratamientos_id_pac	tratamientos	pacientes

3. Oficial de seguridad de la base de datos

Tiene como función:

- Se encarga de la creación de roles y privilegios.

```
-- Crear roles
CREATE ROLE auditor NOLOGIN; -- Ya esta creado
CREATE ROLE medico LOGIN PASSWORD 'medico123';
-- Asignar permisos
GRANT SELECT ON pacientes TO auditor;
GRANT SELECT, INSERT, UPDATE ON tratamientos TO medico;
```

- Revisa los log de actividades.

```
-- Registro de actividades (log_statement en postgresql.conf):
log_statement = 'all' -- para auditar cada consulta
```

- Protege los datos mediante encriptacion y cifrado de datos sensibles.

– MD5 hash

```
SELECT md5('contraseña123');
```

– SHA256

```
SELECT encode(digest('contraseña123', 'sha256'), 'hex');
```

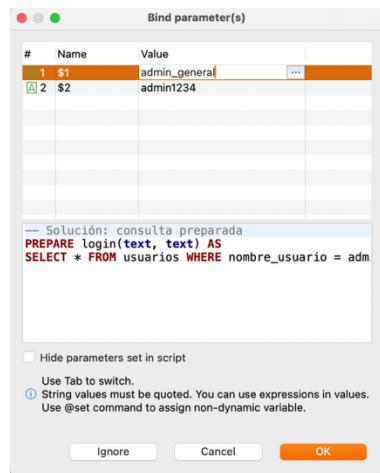
- Simula y mitiga las inyecciones SQL.

– Simulación insegura:

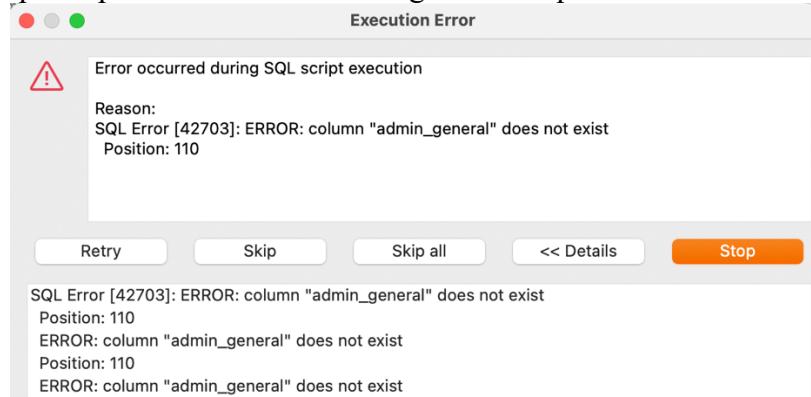
```
SELECT * FROM usuarios WHERE nombre_usuario = 'admin' AND clave =
= '' OR '1'='1';
```

– Solución: consulta preparada

```
PREPARE login(text, text) AS
SELECT * FROM usuarios WHERE nombre_usuario = $1 AND clave =
digest($2, 'sha256');
EXECUTE login('admin_general', 'admin1234');
SELECT * FROM usuarios;
```



Como se puede observar se intenta ingresar datos probando a ver si uno es el que existe en la base para poder vulnerarla, pero con esa consulta preparada se lo puede impedir, ya que los datos realmente existe pero debido a la seguridad que implementa el oficial de seguridad lo que evita el acceso a datos sensibles:



4. Desarrollador de Consultas

Tiene como funciones principales:

- Creación de vistas optimizadas.
- Procedimientos y funciones reutilizables.
- Generación de reportes dinámicos.

— Vista para reportes:

```

CREATE OR REPLACE VIEW vista_resumen_citas AS
SELECT m.nombres AS medico, COUNT(*) AS total_citas
FROM citas_medicas c
JOIN medicos m ON c.id_medico = m.id_medico
GROUP BY m.nombres;

SELECT * FROM vista_resumen_citas;

```

vista_resumen_citas 1 ×

	AZ medico	123 total_citas
1	Carlos	100
2	Maria	99
3	Ana	101
4	Jorge	99
5	Luis	107

Se creó un procedimiento reutilizable para contar los pacientes por un rango de fechas determinado:

```

SELECT total_pacientes_rango('2025-01-01', '2025-08-03');

```

Results 1 ×

	123 total_pacientes_rango
1	6

5. Usuario Final

Tiene como función principal:

- Solo consultas básicas.
- Acceso controlado a vistas controladas.
- No tiene acceso directo a las tablas.

-- Acceso seguro:

-- Crear usuario final

```
CREATE ROLE recepcionista LOGIN PASSWORD 'recepcion123';
```

-- Dar acceso solo a vistas

```
GRANT SELECT ON vista_resumen_citas TO recepcionista;
```

-- El usuario puede ejecutar:

```
SELECT * FROM vista_resumen_citas;
```

-- Pero no puede ver ni modificar la tabla original citas.

Ejemplo de vista del cliente:

	A-Z medico	total_citas
1	Carlos	100
2	María	99
3	Ana	101
4	Jorge	99
5	Luis	107

Respaldo y Restauración

- BACKUP en Frío (Servidor detenido)

Paso 1: Detener el servicio de PostgreSQL:

```
docker stop postgres_ssl
```

Paso 2: Copiar los datos y enviarlos a la carpeta de respaldos en frío:

```
docker cp postgres_ssl:var/lib/postgresql/data ~/respaldos/frio/backup_frio.sql
```

Paso 3: Verificar si se movieron los datos de respaldo:

```
Successfully copied 67.2MB to /Users/adrian_ramos/respaldos/frio/backup_frio.sql
```

Aquí se puede ver que la carpeta contenedora sí tiene los archivos de respaldo:

Nombre	Fecha de modificación	Tamaño	Clase
backup_frio.sql	hoy, 14:32	--	Carpetas
> base	ayer, 11:57	--	Carpetas
> global	hoy, 14:11	--	Carpetas
> pg_commit_ts	ayer, 11:57	--	Carpetas
> pg_dynshmem	ayer, 11:57	--	Carpetas
pg_hba.conf	ayer, 11:57	6 KB	Documento de Texto
pg_ident.conf	ayer, 11:57	3 KB	Documento de Texto
> pg_logical	hoy, 14:29	--	Carpetas
> pg_multixact	ayer, 11:57	--	Carpetas
> pg_notify	ayer, 11:57	--	Carpetas
> pg_replslot	ayer, 11:57	--	Carpetas
> pg_serial	ayer, 11:57	--	Carpetas
> pg_snapshots	ayer, 11:57	--	Carpetas
> pg_stat	hoy, 14:29	--	Carpetas
> pg_stat_tmp	hoy, 14:29	--	Carpetas
> pg_subtrans	ayer, 11:57	--	Carpetas
> pg_tblspc	ayer, 11:57	--	Carpetas
pg_twophase	ayer, 11:57	--	Carpetas

- BACKUP en Caliente (Servidor activo)

Ingresamos a la terminal:

Paso 1: Crear la carpeta dentro del contenedor

```
mkdir -p /respaldos/caliente
```

Paso 2: Ejecutar el respaldo en caliente

```
pg_dump -U postgres -d ssl_test > /respaldos/caliente/backup-caliente.sql
```

Paso 3: Verificar que se creó el archivo

```
ls -lh /respaldos/caliente
```

Como se puede ver que si existe el respaldo realizado en caliente:

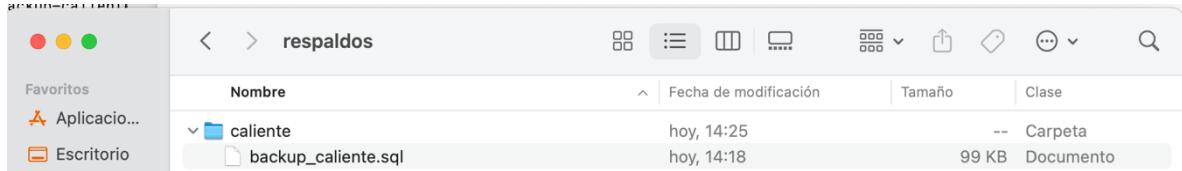
```
[root@83e620912fa3:/# ls -lh /respaldos/caliente
total 100K
-rw-r--r-- 1 root root 97K Aug  2 19:18 backup_caliente.sql
root@83e620912fa3:/# ]
```

Luego se procede a copiar y pegar el archivo de respaldo para tenerlo en un directorio accesible:

Primero se sale del contenedor y se usa docker cp para copiar el archivo:

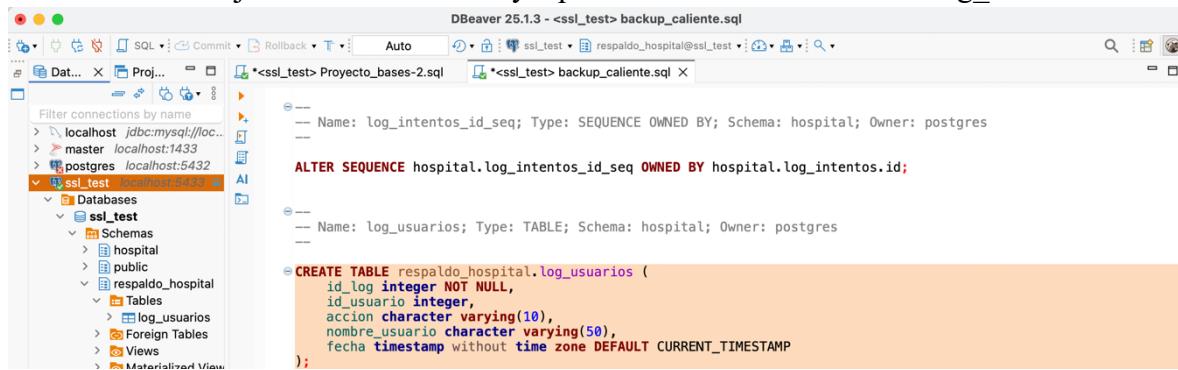
```
docker cp postgres_ssl:/respaldos/caliente/backup_caliente.sql ~/Documents/backup-caliente.sql
```

```
(base) adrian_ramos@MacBook-Air-de-Adrian ~ % docker cp postgres_ssl:/respaldos/caliente/backup_caliente.sql ~/respaldos/backup_caliente.sql
Successfully copied 101kB to /Users/adrian_ramos/respaldos/backup_caliente.sql
(base) adrian_ramos@MacBook-Air-de-Adrian ~ %
```



Restauración parcial o completa

Luego de realizar los diferentes respaldos, se procedio a abrir el script backup_caliente.sql en el entorno de ejecución de DBeaver y a probar restaurar una tabla de log_usuarios:



Como se puede evidenciar cargo el scrip de respaldo en postgres para luego crear un nuevo esquema llamado respaldo_hospital y por ultimo restaurar una tabla.