Proyecto final - Gestión de veterinarios

Javier Gómez Alayón - alu0101562445

Yoel Benítez Cabrera - alu0101592305

Airam Prieto González - alu0101546377

Objetivos del proyecto

El propósito del proyecto es diseñar una base de datos que permita administrar de manera eficiente la información relacionada con la atención veterinaria de mascotas, incluyendo la gestión de pacientes, dueños, personal veterinario, tratamientos, citas, facturación y medicamentos.
El sistema busca: facilitar el seguimiento médico de cada mascota, así como permitir la coordinación entre distintas clínicas de la red. Ofrecer una garantía sobre la trazabilidad de medicamentos y tratamientos aplicados.

El objetivo principal de este proyecto es el diseño, implementación y despliegue de un sistema de información robusto para la gestión integral de una red de clínicas veterinarias.

Desde la perspectiva operativa, el sistema busca solucionar la problemática de la gestión descentralizada y la falta de integridad en los datos clínicos. Los objetivos específicos son: Gestión Centralizada y Eficiente: (administrar de manera unificada la información crítica de la red de clínicas), trazabilidad Clínica (garantizar un seguimiento médico exhaustivo de cada mascota, manteniendo un historial único que refleje tratamientos, diagnósticos y evolución) y control de Inventario y Seguridad: (asegurar la trazabilidad completa de los medicamentos suministrados en cada tratamiento).

Contexto y especificación de requisitos

El sistema propuesto se crea para solucionar los problemas de gestión sobre una red preestablecida de veterinarios de una misma zona geográfica, que usaban un sistema obsoleto de gestión. El nuevo sistema propone una base de datos centralizada que dará servicio a todas las clínicas de la red. En este entorno, los clientes pueden acudir a cualquiera de las sedes, y su información, así como la de sus mascotas. Debe estar disponible inmediatamente para el personal sanitario, independientemente de dónde se haya realizado el alta inicial. La operativa abarca desde la gestión administrativa (citas y facturación) hasta la estrictamente clínica (diagnósticos y tratamientos).

Desde la perspectiva de la ingeniería de datos, el proyecto tiene como finalidad demostrar la capacidad de modelar escenarios complejos y desarrollar soluciones escalables utilizando tecnologías estándares de la industria:

- **Modelado semántico avanzado** Diseñar un esquema conceptual (Modelo E/R) que capture fielmente la realidad del negocio utilizando estructuras complejas y, de esta manera, poder hacer a la base de datos y su funcionamiento fácil de ver y entender de un primer vistazo.
- **Implementación en PostgreSQL** Materializar el diseño en el Sistema Gestor de Bases de Datos PostgreSQL, asegurando la integridad de los datos mediante el uso de claves foráneas, restricciones CHECK y la programación de disparadores (triggers) y funciones almacenadas para validar reglas de negocio complejas (como la incompatibilidad de cargos o validaciones temporales).

- **Desarrollo de Interfaz de Acceso (API)** Implementar una API REST funcional utilizando Flask, que exponga una capa de abstracción segura para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre la base de datos, facilitando la futura integración con aplicaciones web o móviles.

Entidades principales

- **Clínica** (id_clínica, nombre, dirección, teléfono)
- **Empleado** (id_empleado, nombre, apellido, cargo, salario, id_clínica)
- **Cliente** (id_cliente, nombre, apellido, teléfono, email)
- **Mascota** (id_mascota, nombre, especie, raza, fecha_nacimiento, id_cliente)
- **Cita** (id_cita, fecha, hora, id_mascota, id_empleado)
- **Tratamiento** (id_tratamiento, descripción, tipo, costo)
- **Medicamento** (id_medicamento, nombre, dosis_recomendada, proveedor)
- **Factura** (id_cliente, id_factura, fecha_emisión, total)

Entidades débiles

Factura (id_cliente, id_factura, fecha_emisión, total) es una entidad débil ya que depende de que existan registros de clientes a los que estar asociados para poder existir

Relación triple

La relación triple de nuestro diseño es la de Empleado, Tratamiento y Mascota, esto nos permitirá relacionar qué empleado le administró qué tratamiento a qué mascota

Jerarquía IS_A

Empleado IS_A
- Veterinario (número_matricula, especialidad)
- Asistente (becado)

Donde cada subentidad hereda los atributos básicos del empleado y adquiere sus respectivos atributos según su cargo

Relaciones 1:N

Una Clínica tiene varios Empleados.

Un Cliente puede tener varias Mascotas.

Una Mascota puede tener varias Citas.

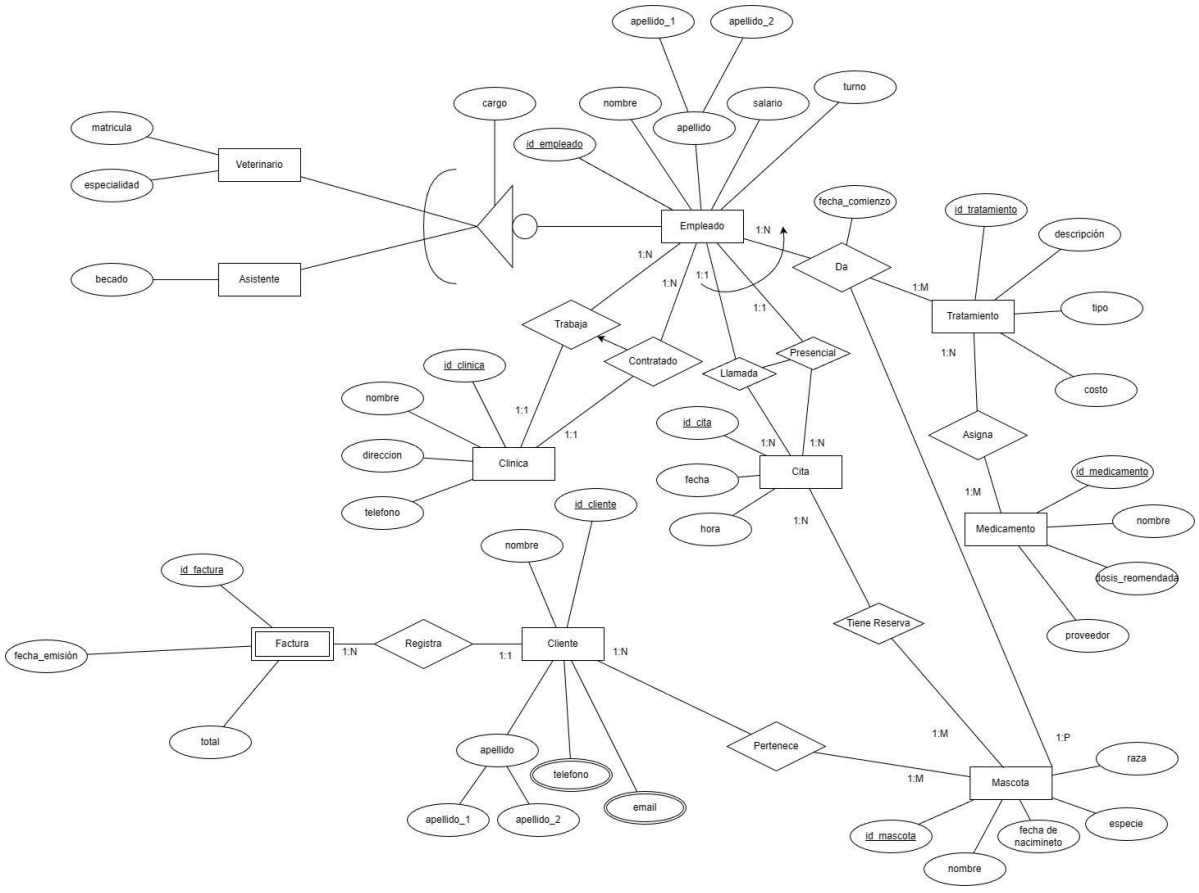
Relaciones M:N

Un Tratamiento puede requerir varios Medicamentos, y un Medicamento puede ser usado en varios Tratamientos, la forma en la que se desarrolla a la hora de implementarlo es Asigna(id_tratamiento, id_medicamento).

Restricciones de inclusión / exclusión / inclusividad / exclusividad

Inclusión: En nuestro diagrama vemos que un Empleado no puede estar relacionado con una Clínica si no está contratado antes.
Exclusión: Un empleado sólo podrá atender una cita o bien de manera telefónica o bien de manera presencial.
Inclusividad: Para que un empleado pueda dar un tratamiento ha tenido primero que atender una cita.
Exclusividad: Un empleado solo podrá ser veterano o asistente pero no los dos a la vez

Modelo Entidad Relación



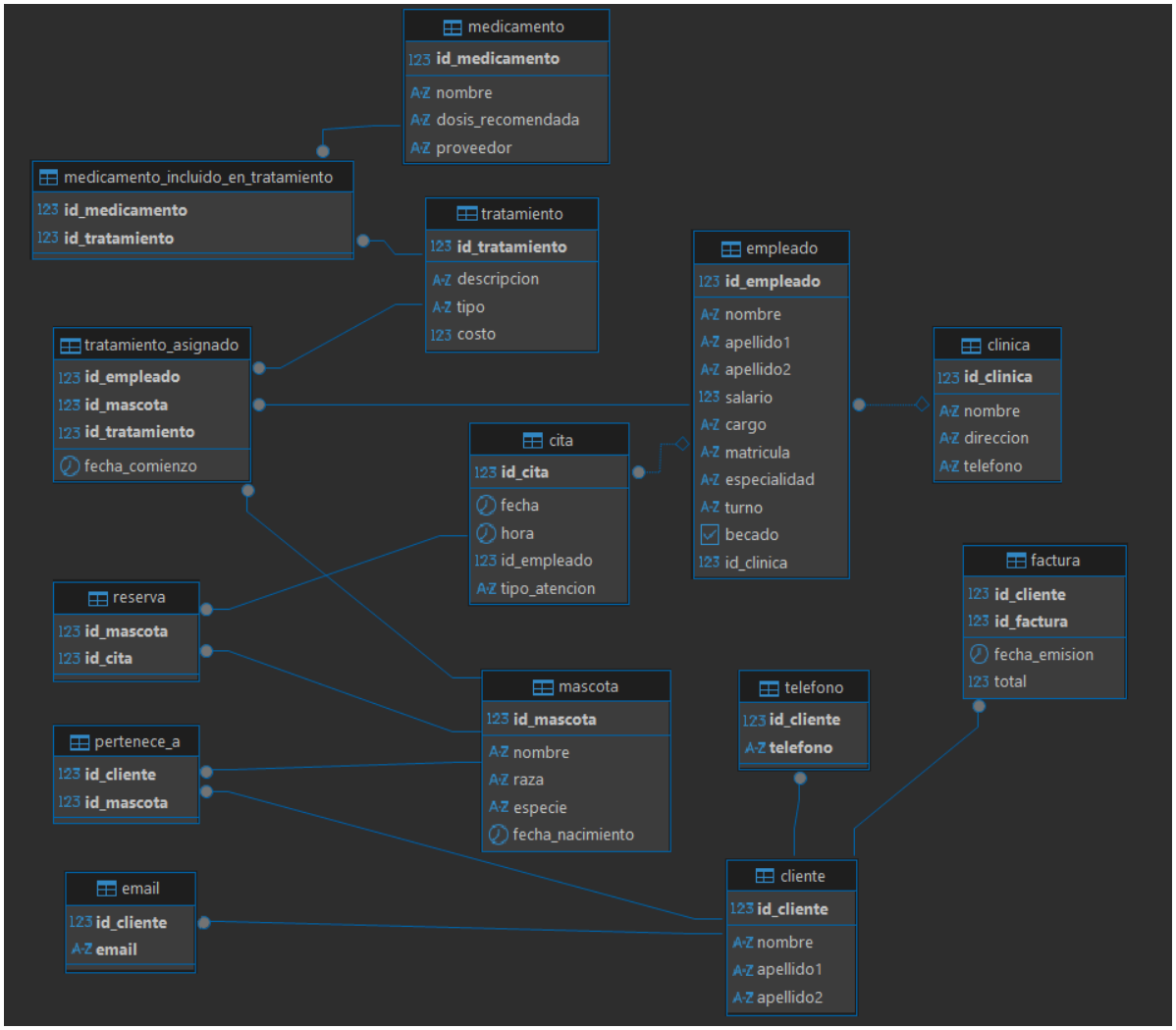
Modelo relacional

Clínica: (id_clinica, nombre, dirección, teléfono, **id_clinica**)
Cita: (id_cita, fecha, hora, **id_empleado,** tipo_atención)
Factura: (id_factura, **id_cliente**, fecha_emisión, total)
Cliente: (id_cliente, nombre, apellido1, apellido2)
Mascota: (id_mascota, nombre, raza, especie, fecha_nacimiento)
Tratamiento: (id_tratamiento, descripción, tipo, costo)
Medicamento: (id_medicamento, nombre, dosis_recomendada, proveedor)

Empleado: (id_empleado, nombre, apellido1, apellido2, salario, cargo, matrícula, especialidad, turno, becado, **id_clinica**)

Tratamiento asignado: (id_empleado, id_mascota, id_tratamiento, fecha_comienzo)
Asigna: (id_medicamento, id_tratamiento)
Pertenece: (id_cliente, id_mascota)
Reserva: (id mascosta, id_cita)

Teléfono: (id_cliente, teléfono)
Email: (id_cliente, email)



Justificaciones

Para solucionar la herencia de la entidad Empleado se optó por incluir los atributos de las subentidades dentro de la entidad principal.

Para diferentes relaciones N:M se crean diferentes entidades que representen esa relación

A la hora de representar atributos multievaluados se crean entidades como Teléfono e Email.

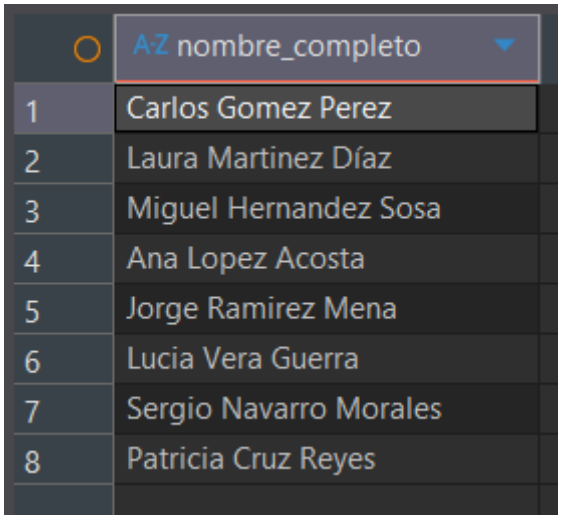
Ejemplos de consultas en sql:

```SQL
-- Listar todos los clientes con su nombre completo
select nombre || ' ' || apellido1 || ' ' || apellido2 AS nombre_completo
FROM Cliente;

-- Listar todas las mascotas con su especie y raza
SELECT nombre, especie, raza
FROM Mascota;


-- Mostrar todas las mascotas con sus dueños
SELECT m.nombre AS mascota, c.nombre AS dueño, c.apellido1, c.apellido2
FROM Mascota m
JOIN Pertenece_a pa ON m.id_mascota = pa.id_mascota
JOIN Cliente c ON pa.id_cliente = c.id_cliente;

-- Mostrar facturas con el nombre completo del cliente
SELECT f.id_factura, f.fecha_emision, f.total,
       c.nombre || ' ' || c.apellido1 || ' ' || c.apellido2 AS cliente
FROM Factura f
JOIN Cliente c ON f.id_cliente = c.id_cliente;

-- Sumar el total de facturas por cliente
SELECT c.nombre || ' ' || c.apellido1 AS cliente, SUM(f.total) AS total_facturas
FROM Cliente c
JOIN Factura f ON c.id_cliente = f.id_cliente
GROUP BY c.id_cliente, c.nombre, c.apellido1
ORDER BY total_facturas DESC;

-- Listar los empleados por clínica
SELECT cl.nombre AS clinica, e.nombre AS empleado, e.cargo, e.turno
FROM Empleado e
JOIN Clinica cl ON e.id_clinica = cl.id_clinica
ORDER BY cl.nombre, e.nombre;

-- Mostrar todos los tratamientos de cada mascota con el empleado que lo aplicó
SELECT m.nombre AS mascota, t.descripcion, t.tipo, t.costo,
```

```sql
       e.nombre || ' ' || e.apellido1 AS empleado
FROM Tratamiento_asignado ta
JOIN Mascota m ON ta.id_mascota = m.id_mascota
JOIN Tratamiento t ON ta.id_tratamiento = t.id_tratamiento
JOIN Empleado e ON ta.id_empleado = e.id_empleado
ORDER BY m.nombre, t.tipo;

-- Listar los medicamentos usados por cada tratamiento
SELECT t.descripcion AS tratamiento, m.nombre AS medicamento, m.dosis_recomendada
FROM Medicamento_incluido_en_tratamiento mit
JOIN Medicamento m ON mit.id_medicamento = m.id_medicamento
JOIN Tratamiento t ON mit.id_tratamiento = t.id_tratamiento
ORDER BY t.descripcion;

-- Facturas emitidas en el último mes con su cliente y total
SELECT f.id_factura, f.fecha_emision, f.total,
       c.nombre || ' ' || c.apellido1 AS cliente
FROM Factura f
JOIN Cliente c ON f.id_cliente = c.id_cliente
WHERE f.fecha_emision >= (CURRENT_DATE - INTERVAL '1 month');

-- Contar el número de citas por empleado y por clínica
SELECT cl.nombre AS clinica, e.nombre || ' ' || e.apellido1 AS empleado,
COUNT(ci.id_cita) AS num_citas
FROM Empleado e
JOIN Clinica cl ON e.id_clinica = cl.id_clinica
LEFT JOIN Cita ci ON e.id_empleado = ci.id_empleado
GROUP BY cl.nombre, e.nombre, e.apellido1
ORDER BY cl.nombre, num_citas DESC;

-- Promedio de costo de tratamientos por tipo
SELECT tipo, AVG(costo) AS costo_promedio
FROM Tratamiento
GROUP BY tipo
ORDER BY costo_promedio DESC;

-- Mascotas con sus tratamientos y los medicamentos usados
SELECT m.nombre AS mascota, t.descripcion AS tratamiento, m2.nombre AS
medicamento
FROM Tratamiento_asignado ta
JOIN Mascota m ON ta.id_mascota = m.id_mascota
JOIN Tratamiento t ON ta.id_tratamiento = t.id_tratamiento
LEFT JOIN Medicamento_incluido_en_tratamiento mit ON t.id_tratamiento =
mit.id_tratamiento
LEFT JOIN Medicamento m2 ON mit.id_medicamento = m2.id_medicamento
ORDER BY m.nombre, t.descripcion;
```

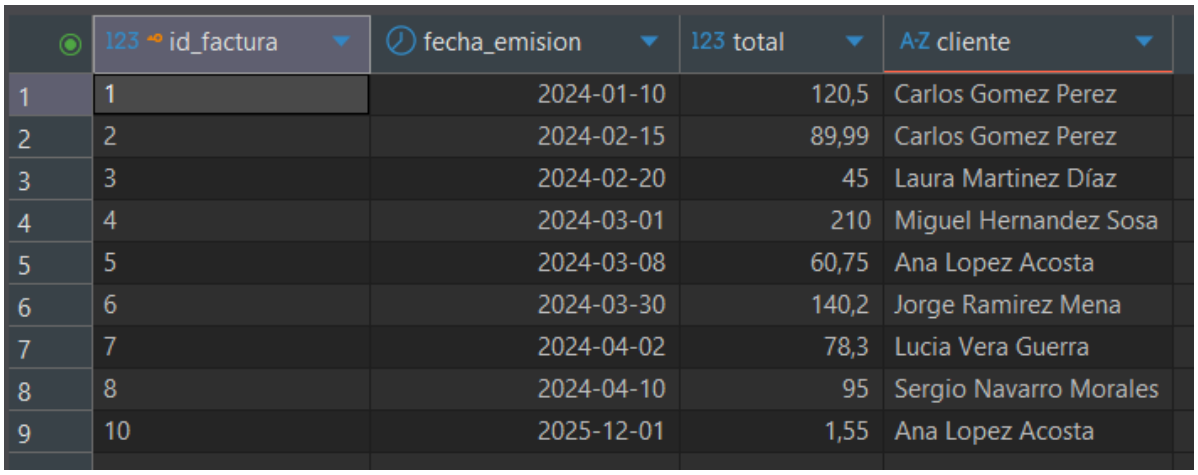
Algunas de las tablas resultantes han sido las siguientes:

Seleccionar todos los clientes



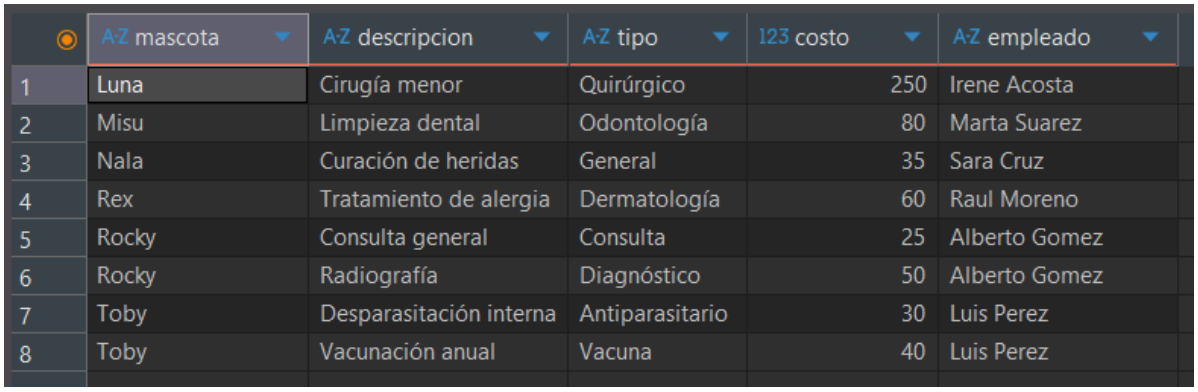
	nombre_completo
1	Carlos Gomez Perez
2	Laura Martinez Díaz
3	Miguel Hernandez Sosa
4	Ana Lopez Acosta
5	Jorge Ramirez Mena
6	Lucia Vera Guerra
7	Sergio Navarro Morales
8	Patricia Cruz Reyes

Mostrar facturas con el nombre completo del cliente



	id_factura	fecha_emision	total	cliente
1	1	2024-01-10	120,5	Carlos Gomez Perez
2	2	2024-02-15	89,99	Carlos Gomez Perez
3	3	2024-02-20	45	Laura Martinez Díaz
4	4	2024-03-01	210	Miguel Hernandez Sosa
5	5	2024-03-08	60,75	Ana Lopez Acosta
6	6	2024-03-30	140,2	Jorge Ramirez Mena
7	7	2024-04-02	78,3	Lucia Vera Guerra
8	8	2024-04-10	95	Sergio Navarro Morales
9	10	2025-12-01	1,55	Ana Lopez Acosta

Mostrar todos los tratamientos de cada mascota con el empleado que lo aplicó



	mascota	descripcion	tipo	costo	empleado
1	Luna	Cirugía menor	Quirúrgico	250	Irene Acosta
2	Misu	Limpieza dental	Odontología	80	Marta Suarez
3	Nala	Curación de heridas	General	35	Sara Cruz
4	Rex	Tratamiento de alergia	Dermatología	60	Raul Moreno
5	Rocky	Consulta general	Consulta	25	Alberto Gomez
6	Rocky	Radiografía	Diagnóstico	50	Alberto Gomez
7	Toby	Desparasitación interna	Antiparasitario	30	Luis Perez
8	Toby	Vacunación anual	Vacuna	40	Luis Perez

Promedio de costo de tratamientos por tipo

	tipo	costo_promedio
1	Quirúrgico	250
2	Odontología	80
3	Dermatología	60
4	Vacuna	50
5	Diagnóstico	50
6	General	35
7	Antiparasitario	30
8	Consulta	25

Mascotas con sus tratamientos y los medicamentos usados

	mascota	tratamiento	medicamento
1	Luna	Cirugía menor	Antibiótico A
2	Luna	Cirugía menor	Sedante H
3	Misu	Limpieza dental	Analgésico D
4	Nala	Curación de heridas	Crema F
5	Rex	Tratamiento de alergia	Antiinflamatorio E
6	Rocky	Consulta general	[NULL]
7	Rocky	Radiografía	[NULL]
8	Toby	Desparasitación interna	Antiparasitario C
9	Toby	Vacunación anual	Vacuna G
10	Toby	Vacunación anual	Vacuna B

Triggers

Hemos puesto un trigger para el manejo inteligente del tipo de empleado que se registra en la base de datos, para una gestión más precisa y unos lanzamientos de errores más concretos según el caso.
Si el empleado es de cargo veterinario no puede estar becado y si es asistente no puede tener matrícula y especialidad.

```sql
-- validación de los subtipos de empleados
CREATE OR REPLACE FUNCTION validar_empleado() RETURNS TRIGGER AS $$
BEGIN
    -- Veterinario no puede estar becado
    IF NEW.cargo ILIKE 'veterinario' AND NEW.becado = TRUE THEN
        RAISE EXCEPTION 'Un empleado veterinario no puede estar becado';
    END IF;

    -- Asistente no puede tener matricula ni especialidad
    IF NEW.cargo ILIKE 'asistente' THEN
        IF NEW.matricula IS NOT NULL THEN
            RAISE EXCEPTION 'Un asistente no puede tener matrícula';
        END IF;
        IF NEW.especialidad IS NOT NULL THEN
            RAISE EXCEPTION 'Un asistente no puede tener especialidad';
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_validar_empleado
BEFORE INSERT OR UPDATE ON Empleado
FOR EACH ROW
EXECUTE FUNCTION validar_empleado();
```

Ejemplos de updates en sql:

```sql

-- Error por el trigger trg_validar_total_factura
UPDATE Factura
SET total = -50
WHERE id_factura = 1;
```

```sql
-- Error por la constraint de validar_fecha_factura
INSERT INTO Factura (fecha_emision, total, id_cliente)
VALUES ('2100-01-01', 100, 1);

-- Error de un trigger: Un veterinario no puede estar becado
UPDATE Empleado
SET becado = TRUE
WHERE cargo ILIKE 'Veterinario' AND id_empleado = 1;

-- Error de un trigger: un asistente no puede tener matrícula
UPDATE Empleado
SET matricula = 'MAT9999'
WHERE cargo ILIKE 'Asistente' AND id_empleado = 2;

-- Error de un trigger: un asistente no puede tener especialidad
UPDATE Empleado
SET especialidad = 'Cirugía'
WHERE cargo ILIKE 'Asistente' AND id_empleado = 2;
```

Los distintos errores que se muestran:

```
SQL Error [23514]: ERROR: new row for relation "factura" violates check
constraint "chk_factura_total_no_negativo"
  Detail: Failing row contains (1, 1, 2024-01-10, -50.00).
```

```
SQL Error [23514]: ERROR: new row for relation "factura" violates
check constraint "chk_factura_fecha"
  Detail: Failing row contains (1, 10, 2100-01-01, 100.00).
```

```
SQL Error [P0001]: ERROR: Un empleado veterinario no puede estar
becado
  Where: PL/pgSQL function validar_empleado() line 5 at RAISE
```

```
SQL Error [P0001]: ERROR: Un asistente no puede tener matrícula
  Where: PL/pgSQL function validar_empleado() line 11 at RAISE
```

```
SQL Error [P0001]: ERROR: Un asistente no puede tener especialidad
  Where: PL/pgSQL function validar_empleado() line 14 at RAISE
```

Ejemplos de “delete” en sql:

```sql
-- Tratamiento 1 (Vacunación anual)
DELETE FROM Tratamiento
WHERE id_tratamiento = 1;
-- Esto hace lo siguiente:
-- 1) Eliminar registros en Tratamiento_asignado que usen este tratamiento
-- 2) Eliminar registros en Medicamento_incluido_en_tratamiento

-- Medicamento 2 (Vacuna B)
DELETE FROM Medicamento
WHERE id_medicamento = 2;
-- Esto hace lo siguiente:
-- 1) Eliminar Medicamento_incluido_en_tratamiento asociado
-- 2) No afecta directamente a Tratamiento_asignado

-- Cliente 1 (Carlos) tiene facturas, teléfonos, emails y mascotas
DELETE FROM Cliente
WHERE id_cliente = 1;
-- Esto hace lo siguiente:
-- 1) Eliminar todas sus facturas automáticamente
-- 2) Eliminar todos sus teléfonos
-- 3) Eliminar todos sus emails
-- 4) Eliminar la relación en Pertenece_a con sus mascotas


-- Empleado 4 (Irene)
DELETE FROM Empleado
WHERE id_empleado = 4;
-- Esto hace lo siguiente:
-- 1) En la tabla Cita, id_empleado se pone a NULL para sus citas
-- 2) En Tratamiento_asignado, sus tratamientos se eliminan por "ON DELETE
CASCADE"
```

Flask API

Para el acceso y consulta de la información almacenada en la base de datos se ha desarrollado una API REST utilizando el framework Flask. Esta API actúa como capa intermedia entre la base de datos PostgreSQL y los posibles clientes (aplicaciones web, scripts u otras herramientas), permitiendo obtener los datos de forma estructurada y segura mediante peticiones HTTP.

La API está orientada exclusivamente a operaciones de consultas (GET), ya que su objetivo principal es facilitar la visualización y análisis de la información del sistema veterinario sin modificar los datos almacenados. Todas las respuestas se devuelven en formato JSON, lo que garantiza la compatibilidad con distintos tipos de clientes y tecnologías.

En el siguiente enlace se podrá ver la implementación de la misma y cómo usarla: Flask API.