TRABAJO FINAL INTEGRADOR

GESTIÓN VETERINARIA

UNIVERSIDAD TECNOLÓGICA NACIONAL

Base de Datos I

Diana Falla - Claudio Fiorito - Jennifer Franco -
Jonathan Franco

EL DESAFÍO: CONSTRUIR UNA BASE SÓLIDA

El objetivo fue diseñar e implementar desde cero una base de datos relacional, normalizada y escalable para gestionar la operatoria completa de una clínica veterinaria.

Funcionalidades clave a soportar:

- Gestión de dueños y profesionales.
- Historial detallado de mascotas (raza, especie, microchip).
- Registro de eventos clave como la implantación de microchips.

ETAPAS

1. Modelado y Definición de Constraints
2. Generación y carga de datos masivos con SQL puro
3. Consultas Avanzadas y Reportes
4. Seguridad e Integridad
5. Concurrencia y Transacciones

ETAPA 1: EL MODELO ENTIDAD-RELACIÓN

ENTIDADES PRINCIPALES

1. VETERINARIA
2. PERSONA
3. MASCOTA
4. MICROCHIP

DECISIONES CLAVE: NORMALIZACIÓN

Personas (Generalización):

Se creó una tabla Persona para unificar los datos de Dueños y Veterinarios, evitando redundancia.

Mascotas (Clasificación):

Se crearon las tablas catálogo Especie y Raza para estandarizar la información y evitar errores.

Ubicaciones (Catálogos):

Se crearon las tablas Provincia y Cod_Postal para no repetir datos de localidad en cada dirección.

LAS REGLAS DEL JUEGO: CONSTRAINTS

Implementamos un conjunto completo de restricciones para proteger los datos

FOREIGN KEY

Forzamos las relaciones y definimos comportamientos como ON DELETE RESTRICT para no perder datos importantes.

UNIQUE

Garantizamos que datos como el DNI, el código del microchip o la matrícula del veterinario no se repitan.

CHECK

Validamos la calidad de los datos, como impedir nombres vacíos (TRIM(nombre) <> '') o verificar formatos de email.

ETAPA 2: GENERACIÓN Y CARGA DE DATOS MASIVOS CON SQL PURO

Uso de tablas semilla y funciones SQL

¿QUÉ SON LAS “TABLAS SEMILLA”?

Son tablas pequeñas que usamos como base para combinar valores y generar miles de registros automáticos

UTILIZAMOS:

- seed_nombres
- seed_apellidos
- seed_calles

Automatizando así la generación de personas, direcciones y emails

¿CÓMO SE MULTIPLICAN LOS DATOS?

Para la carga masiva, combinamos funciones clave con consultas SELECT

INSERT ... SELECT

Sirve para llenar otras tablas usando combinaciones en vez de cargar a manual.

CONCAT()

Se usa para generar cadenas de texto dinámicas y únicas. Por ejemplo: CONCAT('user', n, '@mail.com') → genera miles de emails distintos.

MOD (%)

Permite repartir datos sin repetirlos, ejemplo:
(n % @CANT_NOMBRES) → elige nombres o apellidos de forma cíclica.

ETAPA 3:

Consultas y generación de información

Con la base llena, el desafío fue sacar valor de la información. Pasamos de tener miles de registros a generar respuestas concretas.

Conectamos las piezas con JOINs
Unimos tablas de veterinarios, mascotas, dueños y microchips para ver cómo se relaciona todo el sistema.

Agrupamos y analizamos con GROUP BY
Usamos funciones como COUNT() y SUM() para calcular totales, promedios y rankings reales

Ordenamos con ORDER BY y creamos rankings
Identificamos los veterinarios con más implantaciones, los microchips activos y dueños que tienen sus mascotas con microchip .

💡 CONCLUSIÓN:

LA ETAPA 3 MOSTRÓ EL PODER DEL SQL PARA TRANSFORMAR DATOS EN INFORMACIÓN SIGNIFICATIVA, DEMOSTRANDO QUE LA ESTRUCTURA CREADA EN LA ETAPA 2 FUNCIONA Y RESPONDE PREGUNTAS REALES DEL CONTEXTO VETERINARIO.

ETAPA 4: SEGURIDAD Y INTEGRIDAD

Objetivos:

- Aplicar la regla de usuario con mínimos privilegios.
- Usar vistas para filtrar y simplificar acceso.
- Respetar integridad referencial y unicidad.
- Aplicar seguridad en el código Java o SQL, según la opción elegida

CREACION DE USUARIO CON MINIMOS PRIVELEGIOS

Se creó un usuario limitado (usuario_consulta) con permisos mínimos, solo de lectura sobre las vistas.
De esta forma no puede insertar, modificar ni eliminar datos, cumpliendo con la regla de privilegios mínimos y reforzando la seguridad del sistema.

DISEÑO DE VISTAS PARA PROTEGER DATOS SENSIBLES

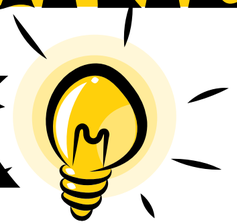
Se crearon vistas para mostrar la información necesaria sin exponer datos privados.

→ **vw_duenios_publico:** oculta DNI, email y teléfono de los dueños.

→ **vw_mascotas_publico:** enmascara el microchip de las mascotas mostrando solo los últimos dígitos.

De esta manera se protege la privacidad de las personas y se restringe el acceso a información crítica.

PRUEBAS DE INTEGRIDAD CON PK Y FK



Se realizaron pruebas de integridad para comprobar que las restricciones funcionen correctamente:

→ Inserción de un ID duplicado en una tabla, que generó error por violación de PK.

→ Inserción de una mascota con un dueño inexistente, que generó error por violación de FK.

De esta forma se asegura la consistencia y validez de los datos en la base.

PROCEDIMIENTO ALMACENADO SEGURO

→ Se creó un procedimiento almacenado (buscar_mascota_por_nombre) que utiliza parámetros en lugar de SQL dinámico.

→ Esto evita ataques de inyección SQL, ya que incluso si se ingresa un texto malicioso, la consulta no expone toda la información de la base.

PROCEDIMIENTO ALMACENADO SEGURO

→ Se creó un procedimiento almacenado (buscar_mascota_por_nombre) que utiliza parámetros en lugar de SQL dinámico.

→ Esto evita ataques de inyección SQL, ya que incluso si se ingresa un texto malicioso, la consulta no expone toda la información de la base.

VERIFICACIÓN DE ACCESO RESTRINGIDO

Se verificó que el usuario limitado (usuario_consulta) solo pueda ejecutar consultas de lectura en las vistas autorizadas.

→ **Acceso permitido:** SELECT en vistas públicas.

→ **Acceso denegado:** intentos de INSERT o DELETE generan error de permisos.

Esto demuestra que la regla de mínimos privilegios se cumple correctamente.

ETAPA 5: CONCURRENCIA Y TRANSACCIONES

EL DESAFÍO: DEADLOCKS

→ **Qué son:** Dos (o más) usuarios se bloquean mutuamente esperando recursos. ¡Ninguno avanza!

→ **Simulación:** Provocamos un deadlock entre dos sesiones actualizando registros en orden inverso.

→ **Resultado:** MySQL detectó el deadlock y canceló una transacción (Error 1213).

SOLUCIÓN: TRANSACCIONES ATÓMICAS + REINTENTO (RETRY)

Para operaciones críticas, usamos Transacciones (Todo o Nada):
O TODAS las operaciones tienen éxito **(COMMIT)**,
o NINGUNA tiene efecto **(ROLLBACK)**.

Implementación (Procedimiento Almacenado):

- **START TRANSACTION / COMMIT / ROLLBACK**
- **Manejo de Deadlocks:**
  - → Detecta el error 1213.
  - → Hace ROLLBACK.
  - → Espera (SLEEP).
  - → Reintenta la transacción (hasta 2 veces)

Resultado: Operaciones seguras y resilientes a fallos por concurrencia.

¿QUÉ "VEN" LAS TRANSACCIONES?: NIVELES DE AISLAMIENTO

READ COMMITTED:

Ve cambios confirmados por otros durante su ejecución (lecturas no repetibles).

REPEATABLE READ (Default):

Mantiene una vista consistente durante toda la transacción (evita lecturas no repetibles)

CONCLUSIÓN

Este proyecto consolidó los conocimientos de **Bases de Datos I**, aplicando progresivamente los conceptos fundamentales:

- **Diseño:** Modelado y definición de restricciones para una base robusta.
- **Testeo (Carga Masiva):** Generación de miles de registros para un escenario realista.
- **Gestión y Seguridad:** Consultas complejas, gestión de usuarios, seguridad y concurrencia.

Se completó un ciclo de desarrollo que simula un **entorno profesional real**

¡MUCHAS GRACIAS!