Trabajo Final Integrador (TFI)

Alumnos

Diana Falla (diana.falla.cba@gmail.com)

Claudio Fiorito (Claudio80.cf@gmail.com)

Jennifer Franco (jennyfranco31.jf@gmail.com)

Jonathan Franco (nahuelfranco7@icloud.com)

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Base de Datos I

Docente Titular

Sergio Neira

Docente Tutor

Diego Lobos

Índice

Introducción

El presente Trabajo Final Integrador documenta el ciclo de vida completo del diseño e implementación de una base de datos relacional para un sistema de gestión de una **clínica veterinaria**. El objetivo principal es construir un modelo de datos robusto, normalizado y optimizado, capaz de manejar un gran volumen de información de manera consistente y segura.

Para simular un entorno de desarrollo profesional, el proyecto se ha estructurado en cinco etapas progresivas:

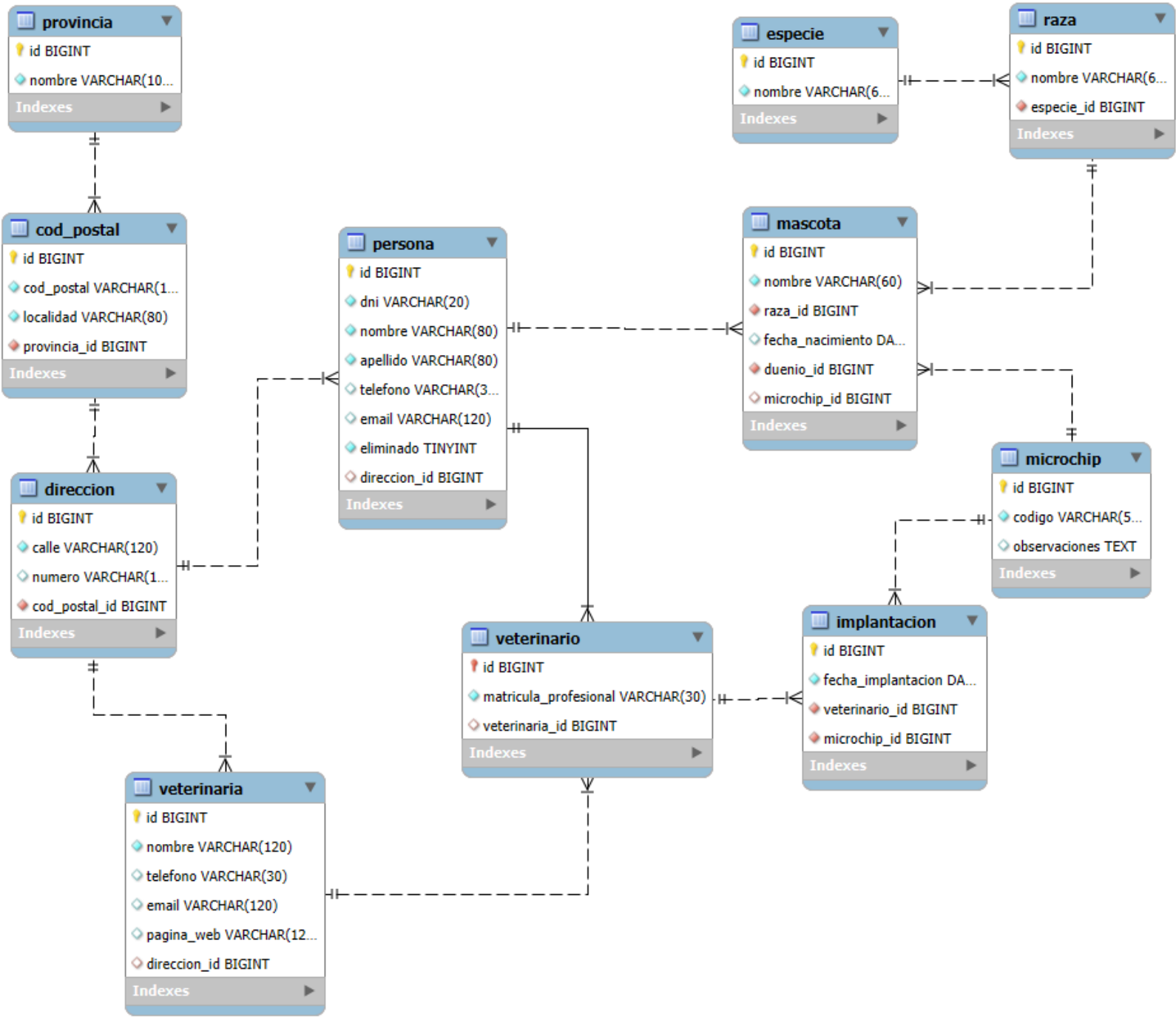
1. **Modelado y Definición de Constraints:** Se establece el diseño conceptual y lógico, definiendo todas las entidades, relaciones y reglas de integridad para garantizar la calidad de los datos desde su origen.
2. **Generación y Carga Masiva de Datos:** Se implementa un script optimizado para poblar la base de datos con un volumen significativo de registros (hasta 35,000 mascotas), creando un entorno realista para las pruebas de rendimiento.
3. **Consultas Avanzadas y Reportes:** Se desarrollan consultas complejas para extraer información de valor, demostrando la utilidad del modelo para generar reportes y análisis.
4. **Seguridad e Integridad:** Se aplican mecanismos de seguridad, como la gestión de usuarios con privilegios mínimos, para proteger la información sensible.
5. **Concurrencia y Transacciones:** Se analizan y gestionan los desafíos del acceso simultáneo a los datos, asegurando la consistencia en operaciones críticas.

A través de estas etapas, se busca consolidar los conocimientos teóricos de la materia aplicándolos a un caso práctico y complejo.

Etapa 1 - Modelado y Definición de Constraints

El proceso de esta etapa abarcó desde el diseño conceptual del DER hasta la creación física de las tablas en SQL (documentada en el script 01_esquema.sql), con un fuerte énfasis en la implementación de un conjunto completo de restricciones para asegurar la integridad de los datos.

Diagrama Entidad-Relación (DER)



Explicación de las Entidades

El modelo de datos se articula en torno a las siguientes entidades, cada una con un propósito específico:

- **Catálogos de Ubicación:** Provincia, Cod_Postal y Direccion.
- **Entidades Principales**: Persona, Mascota, Veterinaria y Microchip.
- **Roles y Especializaciones**: Veterinario (como rol de Persona).

- **Catálogos de Clasificación:** Especie y Raza.
- **Entidades de Eventos**: Implantacion.

Definición de Constraints

Para materializar las reglas del negocio y garantizar la integridad de los datos a nivel de base de datos, se implementó un conjunto de restricciones SQL. Estas constraints son la primera línea de defensa contra datos inconsistentes o inválidos.

- **PRIMARY KEY:** Todas las tablas cuentan con una clave primaria para identificar unívocamente cada registro. La estrategia general fue utilizar una columna id BIGINT AUTO_INCREMENT. La única excepción es la tabla veterinario, cuya PRIMARY KEY (id BIGINT) no es auto-incremental, ya que su valor se hereda directamente de la tabla persona para implementar correctamente el patrón de Generalización.

- **FOREIGN KEY:** Se utilizaron para forzar la integridad referencial entre las tablas. Se definieron comportamientos específicos ON DELETE y ON UPDATE para reflejar con precisión la lógica del negocio:

  - **ON DELETE RESTRICT:** Fue la política más utilizada (ej. entre mascota y persona) para proteger datos maestros, impidiendo el borrado de una persona si todavía tiene mascotas asociadas.

  - **ON DELETE CASCADE:** Se aplicó en relaciones de dependencia existencial fuerte. Por ejemplo, en la relación entre persona y veterinario, al borrar una persona, su rol de veterinario asociado se elimina automáticamente en cascada. Lo mismo ocurre entre microchip e implantacion.

  - **ON DELETE SET NULL**: Se usó en la relación opcional entre mascota y microchip. Si un registro de microchip es eliminado, el campo microchip_id en la mascota asociada se establece en NULL, indicando que la mascota ahora no tiene un chip, pero sin eliminar el registro de la mascota.

  - **ON UPDATE CASCADE:** Implementado en tablas de catálogo como raza, asegura que si el id de una especie llegara a cambiar, la referencia en la tabla raza se actualizaría automáticamente.

- **UNIQUE:** Esta restricción fue clave para garantizar la unicidad de identificadores de negocio y para implementar relaciones uno a uno:

  - Claves Únicas Simples: Se aplicaron a columnas que no pueden repetirse en todo el sistema, como el dni en persona, el codigo en microchip y la matricula_profesional en veterinario.

  - Claves Únicas Compuestas: Se implementó una restricción UNIQUE compuesta en la tabla raza sobre las columnas (nombre, especie_id). Esto impide registrar la

misma raza dos veces para una misma especie (ej. no pueden existir dos "Labrador" para la especie "Perro"), pero sí permite un "Labrador" para "Perro" y un "Labrador" para otra especie, si el caso lo requiriera.

- Relaciones 1 a 1: Se usó UNIQUE en las claves foráneas mascota.microchip_id e implantacion.microchip_id para garantizar que un microchip solo puede estar asociado a una mascota y a un único evento de implantación.

- **CHECK:** Se utilizó sistemáticamente para validar la calidad de los datos a nivel de columna:

  - **Prevención de Cadenas Vacías:** En la mayoría de las tablas se aplicó la restricción CHECK (TRIM(nombre) <> '') para asegurar que los campos de nombres no contengan únicamente espacios en blanco.

  - **Validación de Formato y Longitud:** En la tabla persona, se usó CHECK para validar un formato básico de email (LIKE '%@%') y una longitud mínima para el teléfono. Estas reglas se hicieron más complejas en la tabla veterinaria, permitiendo valores nulos (telefono IS NULL OR ...), lo que demuestra una lógica de negocio más flexible.

Toda esta lógica de constraints (PK, FK, UNIQUE y CHECK) fue implementada en el script 01_esquema.sql. Además, se realizaron pruebas de validación con inserciones y borrados erróneos (contenidas en 02_catalogos.sql) para corroborar que el motor de base de datos rechaza correctamente los datos que violan estas reglas.

Normalización Aplicada

El diseño de la base de datos se adhirió estrictamente a los principios de normalización, principalmente hasta la Tercera Forma Normal (3FN), para eliminar la redundancia de datos y prevenir anomalías de actualización, inserción y borrado.

Las violaciones a las formas normales que existían en un modelo inicial y cómo se solucionaron fueron las siguientes:

1. **Normalización de Direcciones:**

   - Violación (2FN): En un diseño preliminar, los atributos de localidad y provincia podían estar directamente en la tabla direccion, dependiendo solo de una parte de la clave candidata (el código postal). Esto es una dependencia parcial.

   - Solución: Se descompuso la información de ubicación en tres entidades: Provincia, Cod_Postal y Direccion. Direccion ahora solo contiene los datos de la calle y el número, y se vincula a Cod_Postal a través de una clave foránea (cod_postal_id). A su vez, Cod_Postal se vincula a Provincia. Esto asegura que el nombre de una provincia o localidad se almacene una sola vez,

eliminando redundancia y posibles inconsistencias.

2. **Generalización de Persona:**

    - Violación (3FN): Un diseño inicial con tablas separadas para Duenio y Veterinario presentaba una alta redundancia. Ambas tablas contenían exactamente las mismas columnas para datos personales (DNI, nombre, apellido, email, teléfono, dirección). Esto no solo duplica datos, sino que crea dependencias transitivas y dificulta el mantenimiento.

    - Solución: Se aplicó el patrón de Generalización/Especialización. Se creó una super-entidad Persona que centraliza todos los atributos comunes. La tabla Veterinario se convirtió en un rol especializado que "hereda" de Persona a través de una relación 1 a 1, usando el id de persona como su propia clave primaria y foránea. Esto garantiza que cada dato personal se almacene en un único lugar.

3. **Normalización de Especie y Raza en Mascota:**

    - Violación (3FN): Inicialmente, la tabla Mascota contenía los campos especie y raza como atributos de texto. Esto representaba una dependencia transitiva: la raza de una mascota depende de su especie, la cual a su vez depende de la mascota.

    - Solución: Se extrajeron estos atributos a dos tablas de catálogo separadas: Especie y Raza. La tabla Raza contiene una clave foránea que apunta a Especie. La tabla Mascota ahora solo necesita una clave foránea (raza_id) para determinar ambos atributos de forma inequívoca. Esta descomposición elimina la dependencia transitiva y asegura que cada especie y raza se defina una única vez en todo el sistema.

Durante el diseño, se evaluó la posibilidad de llevar la normalización un paso más allá, creando una tabla genérica Contacto para centralizar los campos telefono y email que se repetían en Persona y Veterinaria. Sin embargo, se priorizó la simplicidad y el rendimiento de las consultas. La creación de una tabla Contacto habría requerido un JOIN adicional cada vez que se necesitara consultar el email de una persona o clínica. Se determinó que la pequeña redundancia de estos campos era un compromiso aceptable para mantener las consultas más directas y eficientes, evitando una complejidad innecesaria para el alcance de este proyecto.

Etapa 2 – Generación y carga de datos masivos con SQL puro

Descripción conceptual del mecanismo de carga masiva

Para generar los datos de la base sin usar archivos externos ni procedimientos almacenados, se trabajó con un enfoque totalmente SQL, dividiendo el proceso en etapas para asegurar volumen, integridad y variedad.

Limpieza del esquema

Se desactivaron temporalmente las claves foráneas (FOREIGN_KEY_CHECKS = 0) para poder realizar TRUNCATE en todas las tablas sin conflictos. Esto permite reiniciar la base rápidamente y garantizar que los IDs comiencen desde cero. Una vez vaciado todo, las claves se reactivaron para mantener la integridad en las inserciones posteriores.

Tablas semilla utilizadas

Se crearon tres tablas semilla pequeñas:

- seed_nombres
- seed_apellidos
- seed_calles

Estas tablas permiten combinar datos reales sin escribir miles de valores manualmente. Además, se guardó la cantidad de registros de cada una en variables (@CANT_*) para usarlas con operador % y generar combinaciones controladas.

Tablas maestras o de catálogo

Antes de los insert masivos se cargaron las tablas de referencia:

- provincia
- especie
- raza

Estas tablas tienen pocos datos y se usan como base para claves foráneas posteriores. Esto asegura que las tablas grandes puedan referenciarlas sin errores.

Generación masiva y controlada de datos

➔ **Códigos postales y direcciones**

Con un WITH RECURSIVE se generaron números consecutivos (1 a 100) para crear códigos postales y luego miles de direcciones. Se combinó:

- calle desde tabla semilla
- número dentro de un rango
- un código postal existente

➔ **Personas (dueños y veterinarios)**

Se insertaron personas usando un contador numérico como base.

Se garantizó:

- DNI único (20000000 + n)
- nombres y apellidos rotados con %
- direcciones válidas dentro del rango generado

➔ **Veterinarios**

De todas las personas generadas, las primeras se insertaron en la tabla veterinario. Se les asignó matrícula automática y se vinculó a una clínica existente.

➔ **Microchips y mascotas**

Primero se generaron los microchips con formato secuencial (CHIP-2025-00000001).
Después se cargaron las mascotas tomando:

- nombres predefinidos
- razas con el operador %
- fechas aleatorias
- dueños existentes (usando IDs reales)
- 70% con microchip y 30% NULL (con %)

Para evitar errores de FK, se usaron valores mínimos (@MIN_DUENIO_ID, @MIN_MICROCHIP_ID) en vez de escribir IDs fijos.

➔ **Implantaciones**

Solo se crearon registros para mascotas con microchip. La fecha se calculó desde la fecha de nacimiento y el veterinario se asignó con % para distribuirlos parejo.

¿Cómo se garantiza integridad y cardinalidades?

- Se vació todo con TRUNCATE para evitar residuos.
- Se trabajó con claves foráneas activadas en el momento de inserción.
- Los IDs se calcularon a partir de valores reales existentes (mínimos).
- Las relaciones se respetaron en este orden:
  catálogos → direcciones → personas → veterinarios → microchips → mascotas → implantaciones.
- El operador % se usó para garantizar distribución sin romper claves.

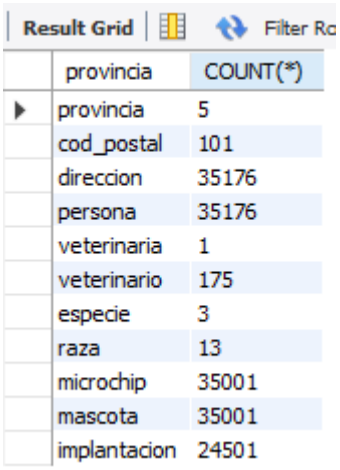
Funciones utilizadas para variedad y consistencia

Se usaron funciones nativas de MySQL para generar datos realistas sin repetir INSERT manual:

- CONCAT
- LPAD
- CASE

- ELT
- DATE_ADD / DATE_SUB
- % (módulo)
- CTE recursivos (WITH RECURSIVE)

Cuadro de verificación (conteos finales)



Result Grid | Filter Ro

provincia	COUNT(*)
provincia	5
cod_postal	101
direccion	35176
persona	35176
veterinaria	1
veterinario	175
especie	3
raza	13
microchip	35001
mascota	35001
implantacion	24501

Etapa 3 – Consultas Avanzadas y Reportes

En esta etapa se desarrollaron consultas SQL avanzadas y reportes que permiten obtener información valiosa del sistema de gestión veterinaria. El objetivo fue analizar y optimizar el rendimiento de la base de datos, identificando áreas donde los reportes aportan valor práctico a la toma de decisiones.

Consultas

A continuación, se describen las consultas creadas, su utilidad, y los resultados obtenidos. Se incluyó además una vista útil para reportes y la implementación de índices para mejorar el rendimiento de búsqueda.
- Ver 05_consultas.sql

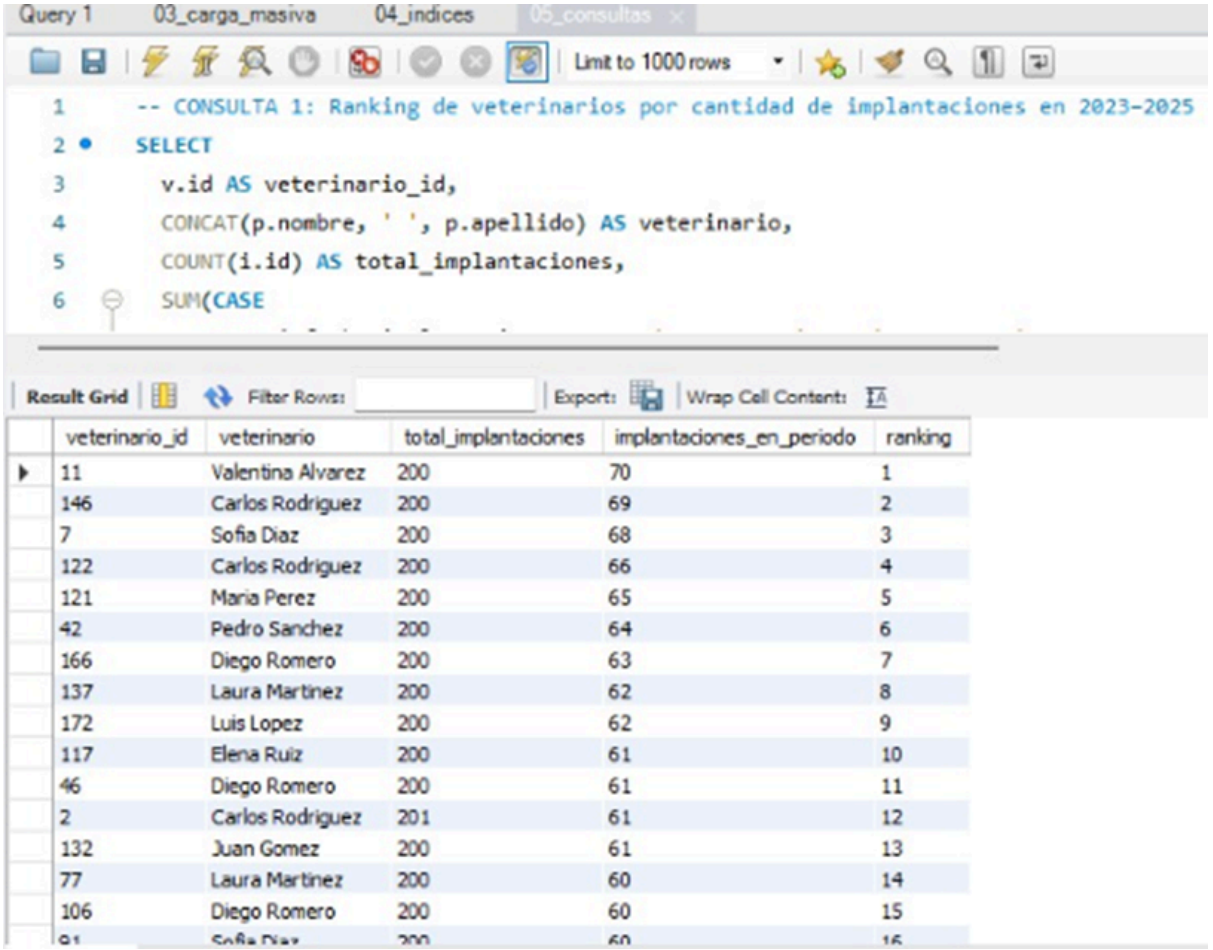
Consulta 1 — Veterinario con más implantaciones (2023–2025)

● Objetivo de la Consulta:
Identificar qué veterinario realizó más implantaciones entre 2023 y 2025, generando un ranking según su nivel de actividad.

● Beneficio para el Negocio:
Esta consulta proporciona una herramienta clave para el análisis del desempeño de los veterinarios dentro del establecimiento. Los resultados permiten reconocer a los profesionales con mayor nivel de actividad, fomentar la mejora continua del equipo y optimizar la asignación de tareas. Además, ayuda a detectar posibles inactividades o

desequilibrios en la carga laboral, lo que facilita una mejor planificación operativa y una gestión más equitativa del personal.

Tipo: JOIN + GROUP BY + ranking.



Query 1 | 03_carga_masiva | 04_indices | 05_consultas

Limit to 1000 rows

```
1   -- CONSULTA 1: Ranking de veterinarios por cantidad de implantaciones en 2023-2025
2   SELECT
3     v.id AS veterinario_id,
4     CONCAT(p.nombre, ' ', p.apellido) AS veterinario,
5     COUNT(i.id) AS total_implantaciones,
6     SUM(CASE
```

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

veterinario_id	veterinario	total_implantaciones	implantaciones_en_periodo	ranking
11	Valentina Alvarez	200	70	1
146	Carlos Rodriguez	200	69	2
7	Sofia Diaz	200	68	3
122	Carlos Rodriguez	200	66	4
121	Maria Perez	200	65	5
42	Pedro Sanchez	200	64	6
166	Diego Romero	200	63	7
137	Laura Martinez	200	62	8
172	Luis Lopez	200	62	9
117	Elena Ruiz	200	61	10
46	Diego Romero	200	61	11
2	Carlos Rodriguez	201	61	12
132	Juan Gomez	200	61	13
77	Laura Martinez	200	60	14
106	Diego Romero	200	60	15
91	Sofia Diaz	200	60	16

Consulta 2 — Listado de Microchips Activos con Mascotas y Propietarios

● **Objetivo de la Consulta:**
Obtener los microchips que permanecen activos (no eliminados) junto con la información detallada de la mascota y su propietario.

● **Beneficio para el Negocio:**
Esta consulta permite mantener un control preciso sobre los microchips en uso, garantizando la trazabilidad de cada mascota registrada. Facilita la identificación de los dueños asociados a cada chip y ayuda a detectar posibles inconsistencias o registros obsoletos en la base de datos. Además, contribuye a una mejor gestión del sistema de identificación animal, favoreciendo la transparencia y la confiabilidad de la información disponible.

```
14    FROM veterinario v
15    JOIN persona p ON v.id = p.id
16    LEFT JOIN implantacion i ON i.veterinario_id = v.id
17    GROUP BY v.id, p.nombre, p.apellido
18    ORDER BY implantaciones_en_periodo DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

microchip_id	codigo	observaciones	eliminado	mascota_id	nombre_mascota	raza	especie	duenio	fecha_implantacion
1	CHIP-2025-00000001	NULL	0	1	Luna	Pastor Alemán	Perro	Miguel Torres	2021-05-20
2	CHIP-2025-00000002	NULL	0	2	Rocky	Bulldog Francés	Perro	Elena Ruiz	2020-04-22
3	CHIP-2025-00000003	NULL	0	3	Lola	Golden Retriever	Perro	Diego Romero	2026-07-30
4	CHIP-2025-00000004	NULL	0	4	Max	Beagle	Perro	Valentina Alvarez	2020-01-26
5	CHIP-2025-00000005	NULL	0	5	Coco	Siamés	Gato	Juan Gomez	2016-08-23
6	CHIP-2025-00000006	NULL	0	6	Toby	Persa	Gato	Maria Perez	2019-01-10
7	CHIP-2025-00000007	NULL	0	NULL	NULL	NULL	NULL	NULL	NULL
8	CHIP-2025-00000008	NULL	0	NULL	NULL	NULL	NULL	NULL	NULL
9	CHIP-2025-00000009	NULL	0	NULL	NULL	NULL	NULL	NULL	NULL
10	CHIP-2025-00000010	NULL	0	10	Bobby	Canario	Ave	Laura Martinez	2025-11-24
11	CHIP-2025-00000011	NULL	0	11	Luna	Periquito	Ave	Pedro Sanchez	2021-09-07
12	CHIP-2025-00000012	NULL	0	12	Rocky	Cacatúa	Ave	Sofia Diaz	2024-01-03
13	CHIP-2025-00000013	NULL	0	13	Lola	Labrador Retrie...	Perro	Miguel Torres	2016-10-12
14	CHIP-2025-00000014	NULL	0	14	Max	Pastor Alemán	Perro	Elena Ruiz	2021-08-06

Consulta 3 — Dueños con 2 o más mascotas registradas

● **Objetivo de la Consulta:**
Identificar a los propietarios que tienen registradas dos o más mascotas dentro del sistema.

● **Beneficio para el Negocio:**
Esta consulta permite detectar a los clientes más fieles y comprometidos con la institución veterinaria, ofreciendo una visión clara de quiénes confían recurrentemente en los servicios brindados. La información obtenida puede utilizarse para implementar estrategias de fidelización, como descuentos, promociones o beneficios especiales. Además, contribuye a mejorar la atención personalizada y a fortalecer el vínculo con los dueños que poseen varias mascotas, promoviendo la satisfacción y la retención de clientes a largo plazo.

Tipo:GROUP BY + HAVING.

```sql
-- CONSULTA 3: Dueños con 2 o más mascotas registradas
SELECT
  p.id AS duenio_id,
  CONCAT(p.nombre, ' ', p.apellido) AS duenio,
  COUNT(m.id) AS num_mascotas,
  GROUP_CONCAT(m.nombre SEPARATOR '; ') AS mascotas_list
FROM persona p
JOIN mascota m ON m.duenio_id = p.id
GROUP BY p.id, p.nombre, p.apellido
HAVING COUNT(m.id) >= 2
```

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

duenio_id	duenio	num_mascotas	mascotas_list

No aparecen registros, lo cual es correcto porque en este momento solo hay dueños con una mascota.

Consulta 4 - Dueños con todas sus mascotas con microchips

● **Objetivo de la Consulta:**
Identificar a los propietarios que ya implantaron un microchip en todas sus mascotas registradas.

● **Beneficio para el Negocio:**
Esta consulta permite realizar un seguimiento preciso del cumplimiento del programa de identificación mediante microchip. Ayuda a reconocer a los dueños más responsables y comprometidos con la trazabilidad y seguridad de sus animales. Además, brinda una base sólida para segmentar campañas informativas o promocionales dirigidas a quienes aún no completaron el proceso, fomentando la regularización y fortaleciendo la gestión integral del sistema de registro.

```sql
-- CONSULTA 4: Dueños con todas sus mascotas microchipeadas
SELECT
 p.id AS duenio_id,
 CONCAT(p.nombre, ' ', p.apellido) AS duenio,
 (SELECT COUNT(*) FROM mascota m2 WHERE m2.duenio_id = p.id) AS total_mascotas
FROM persona p
WHERE NOT EXISTS (
 SELECT 1
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

duenio_id	duenio	total_mascotas
687	Ana Fernandez	1
668	Miguel Torres	1
667	Sofia Diaz	1
666	Pedro Sanchez	1
665	Laura Martinez	1
661	Maria Perez	1
660	Juan Gomez	1
659	Valentina Alvarez	1
658	Diego Romero	1
657	Elena Ruiz	1
656	Miguel Torres	1
669	Elena Ruiz	1

Vista

Objetivo:Generar una vista unificada para consultas y reportes.
Utilidad:Permite acceder rápidamente a información completa sobre implantaciones activas, veterinarios, mascotas y dueños.

- ***Ver 06_vista.sql***

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

implantacion_id	fecha_implantacion	veterinario_id	veterinario	microchip_id	microchip_codigo	n
1	2021-09-04	2	Carlos Rodriguez	1	CHIP-2025-00000001	1
2	2024-12-18	3	Ana Fernandez	2	CHIP-2025-00000002	2
3	2022-12-18	4	Luis Lopez	3	CHIP-2025-00000003	3
4	2022-04-05	5	Laura Martinez	4	CHIP-2025-00000004	4
5	2026-02-11	6	Pedro Sanchez	5	CHIP-2025-00000005	5
6	2017-05-18	7	Sofia Diaz	6	CHIP-2025-00000006	6
7	2018-05-02	11	Valentina Alvarez	10	CHIP-2025-00000010	10
8	2015-09-11	12	Juan Gomez	11	CHIP-2025-00000011	11
9	2018-02-14	13	Maria Perez	12	CHIP-2025-00000012	12

Índices Y Medición De Rendimiento

Para optimizar las consultas, se crearon los siguientes índices:

- ***Ver 04_indices.sql***

- idx_microchip_codigo → búsqueda por código (igualdad)
- idx_implantacion_fecha → búsqueda por rango de fechas
- idx_implantacion_vet → mejora en uniones entre implantación y veterinario

Se realizaron mediciones con y sin índices en tres tipos de consultas: por igualdad, por rango de fechas y con JOIN. Los resultados mostraron que el uso de índices mejora significativamente el rendimiento, especialmente en búsquedas por rango y uniones.

- ***Ver 05_explain-medicion.sql***

RESULTADOS OBTENIDOS:

tipo_consulta	con/sin índice	corrida1_ms	corrida2_ms	corrida3_ms	mediana_ms
igualdad_codigo	con índice	60.418	59.187	61.025	60.418
igualdad_codigo	sin índice	72.350	65.240	70.870	70.870
rango_fechas	con índice	45.600	50.912	48.332	48.332
rango_fechas	sin índice	98.336	64.950	82.867	82.867
join_vet_impl	con índice	75.897	82.937	84.642	82.937
join_vet_impl	sin índice	96.136	88.160	91.436	91.436

En conclusión, el uso de índices y consultas optimizadas demostró ser clave para mejorar el rendimiento de la base de datos. Las consultas avanzadas diseñadas no solo permiten generar reportes útiles para la gestión veterinaria, sino que también muestran cómo un modelo bien estructurado puede escalar eficientemente con grandes volúmenes de datos. Esta etapa evidenció la importancia de combinar un buen diseño lógico con optimización a nivel físico del motor SQL

Etapa 4 – Seguridad e Integridad

En esta etapa se implementaron mecanismos de seguridad y pruebas de integridad en la base de datos **gestion_veterinaria**.

Creación de un usuario con privilegios mínimos

Se creó un usuario denominado **`usuario_consulta`** con contraseña segura en el archivo . Este usuario no posee acceso directo a las tablas, sino únicamente permisos de lectura (SELECT) sobre las vistas definidas.
Esto garantiza el principio de mínimo privilegio, limitando la exposición de información sensible y evitando modificaciones no autorizadas.

- ***Ver 07_seguridad_e_integridad.sql***

Vistas para ocultar datos sensibles

Se implementaron dos vistas:

- **`vw_duenios_publico`**: muestra los datos básicos del dueño y su dirección, pero omite campos sensibles como DNI, email y teléfono.

- **`vw_mascotas_publico`**: lista las mascotas junto a su raza, especie y dueño, enmascarando el número de microchip para evitar exponerlo completo.

  De esta forma, la información relevante puede ser consultada sin comprometer datos privados.

Pruebas de integridad

Se realizaron pruebas de inserciones inválidas para verificar el cumplimiento de las restricciones:

- **Clave primaria/UNIQUE duplicada**: al insertar un registro en `persona` con un DNI repetido, MySQL devolvió el **Error 1062 (Duplicate entry)**, demostrando que la restricción de unicidad está activa.

- **Clave foránea inválida**: al intentar registrar una mascota con un `duenio_id` inexistente, el motor arrojó el **Error 1452 (Cannot add or update a child row: foreign key constraint fails)**, confirmando la integridad referencial.

Estos resultados evidencian que las restricciones PK, UNIQUE y FK protegen la consistencia de los datos.

Procedimiento almacenado seguro (anti-inyección)

Se creó el procedimiento **`buscar_mascota_por_nombre`**, el cual recibe como parámetro el nombre de una mascota y devuelve sus datos asociados.
El procedimiento no utiliza SQL dinámico, sino que trabaja con parámetros, lo que lo hace inmune a intentos de inyección SQL.

- **Prueba normal**: `CALL buscar_mascota_por_nombre('Luna');` devuelve el registro si existe.

- **Prueba maliciosa**: `CALL buscar_mascota_por_nombre("' OR '1'='1");` no devuelve todas las filas, sino que busca literalmente ese texto, evidenciando que no hay vulnerabilidad de inyección.

Pruebas de acceso restringido

Al conectarse con el usuario `usuario_consulta`:

- **Operación permitida**: consultas sobre las vistas, como `SELECT * FROM vw_mascotas_publico`, devuelven datos sin error.

- **Operación denegada**: intentos de inserción o consultas directas sobre tablas, por ejemplo:

```
-- Incorrecto (debe fallar):
INSERT INTO persona (dni,nombre,apellido) VALUES ('1111','X','Y');
-- -> ERROR 1062 (duplicate entry "1" for key "PRIMARY")
```

Etapa 5 – Concurrencia y Transacciones

En esta etapa trabajamos la concurrencia en bases de datos, simulando situaciones reales donde dos usuarios modifican la misma información al mismo tiempo. Todas las pruebas se realizaron con dos sesiones abiertas en MySQL, trabajando sobre la tabla **mascota**, usando transacciones explícitas.
Los ejercicios se implementaron en los scripts **09_concurrencia_guiada.sql** y **08_transacciones.sql**, que contienen las pruebas y el procedimiento de control, respectivamente..

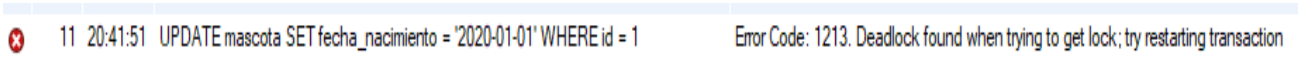
Simulación de bloqueo y deadlock (*script: 09_concurrencia_guiada.sql*

Primero se generó un bloqueo intencional para observar cómo MySQL reacciona cuando dos sesiones compiten por los mismos datos.

En la **Sesión 1** se realizó una actualización sobre un registro de la tabla *mascota* sin ejecutar **COMMIT** ni **ROLLBACK**, lo que dejó el registro bloqueado.

```
9  START TRANSACTION; -- Inicia una operación
10 UPDATE mascota SET nombre = 'Mascota Bloqueo 1' WHERE id = 1; -- Bloquea la mascota 1
```

Luego, en la **Sesión 2**, se inició otra transacción e intentó modificarse otro dato. El conflicto se produce cuando ambas sesiones cruzan sus actualizaciones sobre filas que la otra tiene bloqueada.
El motor detecta un *deadlock* y aborta una de las operaciones con el error **1213 (Deadlock found when trying to get lock)**.
Esto demuestra cómo el sistema protege la integridad de los datos frente a competencia simultánea.



Transacción con retry automática (*script: 08_transacciones.sql*)

Para evitar que una operación crítica se pierda en caso de deadlock, se creó el procedimiento almacenado sp_transferir_mascota. Este contiene:

START TRANSACTION

COMMIT y ROLLBACK

Un manejador que captura el error 1213

Un contador de intentos (retry_count)

Dos reintentos automáticos con SLEEP() si ocurre un bloqueo

Si la operación se ejecuta sin conflicto, se confirma con COMMIT. Si aparece el deadlock, se hace ROLLBACK y se reintenta. Solo si falla más de dos veces informa el error. Esto cumple con el requisito de manejo de concurrencia y recuperación automática sin perder la transacción.

Comparación de niveles de aislamiento(*script: 09_concurrencia_guiada.sql*

Por último, se probó cómo cambia la visibilidad de los datos según el nivel configurado en la sesión.

READ COMMITTED:
La misma transacción puede ver los cambios que otra sesión haya confirmado con COMMIT. Esto se comprobó leyendo el nombre de una mascota, modificándolo desde otra sesión y volviendo a consultar.

REPEATABLE READ:
La transacción mantiene la primera versión leída, aunque otra sesión confirme un cambio mientras tanto. La segunda lectura dentro de la misma transacción devuelve el mismo valor inicial.

Gracias a estas pruebas, se pudo observar claramente la diferencia entre ambos niveles y cómo afectan la lectura concurrente.

Observaciones sobre Concurrencia y Transacciones

Durante esta etapa se comprobó el funcionamiento de la concurrencia en MySQL y cómo el motor garantiza la consistencia ante accesos simultáneos. La simulación del **deadlock** permitió observar que, cuando dos transacciones intentan modificar los mismos datos, el sistema detecta el conflicto y cancela una de ellas para evitar inconsistencias. La **transacción con retry** mostró una estrategia efectiva para mantener la operación segura: reintentar automáticamente tras un error 1213 sin perder los cambios. Finalmente, al comparar los niveles de aislamiento, se evidenció que **READ COMMITTED** permite ver los cambios confirmados por otras sesiones, mientras que **REPEATABLE READ** conserva la misma vista de los datos dentro de una transacción. Estas pruebas reflejan la importancia de definir un buen nivel de aislamiento y de manejar correctamente las transacciones para preservar la integridad y coherencia de la base.

Conclusión

A lo largo de este Trabajo Final Integrador, hemos consolidado los conocimientos de Bases de Datos I, nos permitió aplicar de manera progresiva los conceptos fundamentales de la materia. Comenzando por el modelado y la definición de restricciones, logramos construir una base de datos robusta. Posteriormente, la enfrentamos a un escenario realista mediante la generación y carga masiva de datos. Finalmente, desarrollamos consultas complejas e implementamos medidas críticas de seguridad y manejo de concurrencia, completando así un ciclo de desarrollo que simula un entorno profesional real.

Video Explicativo

Este video documenta la exposición de los alumnos sobre el Trabajo Final Integrador de Bases de Datos I. Los estudiantes explican los temas propuestos y el desarrollo del proyecto, cuyo objetivo es consolidar los contenidos de la asignatura en un entorno

simulado. La presentación se acompaña de una breve demostración que ilustra el funcionamiento de la base de datos.

Link del video en la plataforma de youtube:https://youtu.be/Slymx88_xrc

Repositorio GitHub

Con el objetivo de centralizar todos los artefactos técnicos del proyecto y facilitar su auditoría y revisión, se ha creado un repositorio en GitHub. Este repositorio contiene la totalidad de los entregables, incluyendo:

- El presente informe final en formato PDF.
- Una carpeta (/sql/) que contiene todos los scripts .sql utilizados para:
    - La creación de tablas y constraints (Etapa 1).
    - La generación y carga masiva de datos (Etapa 2).
    - Las consultas, vistas y procedimientos (Etapas 3, 4 y 5).
- El Diagrama Entidad-Relación (DER) del modelo.

Link del repositorio: https://github.com/JenniferFranco/tfi-bd1-utn-veterinaria