Trabajo Final Integrador (TFI)

Aplicación Java con relación 1⟶1 unidireccional + DAO + MySQL

Sistema de Gestión de Veterinaria: Mascotas y Microchips

Integrantes

Gutierrez David

Gutierrez Colque Brian

Iturralde Elortegui Rosario

Guirin Veronica

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Programación II

17 de Noviembre de 2025

Tabla de contenido

Trabajo Final Integrador (TFI)

Objetivos

Desarrollar una aplicación en Java que modele dos clases relacionadas mediante una asociación unidireccional 1 a 1 (la clase “A” referencia a la clase “B”), persistiendo datos en una base relacional mediante JDBC y el patrón DAO, con operaciones transaccionales (commit/rollback) y menú de consola para CRUD.

Consignas generales

• Lenguaje: Java (recomendado 21).
• Persistencia: JDBC (sin ORM) con MySQL.
• Patrón de diseño: DAO y capa Service.
• Código limpio, legible, con manejo de excepciones en todas las capas.
• Relación 1→1 unidireccional: sólo la clase “A” contiene el atributo que referencia a “B”.
• Equipos de 4 personas, sin excepción.

Integrantes y Roles:

- **Brian Gutiérrez Colque**
- **María del Rosario Margarita Iturralde Elortegui**
- **David Gutiérrez**
- **Verónica Guirin**

El trabajo se desarrolló de manera colectiva, con todos los miembros participando de forma integral en las mismas tareas y responsabilidades, sin una división estricta de roles.

Elección del Dominio y Justificación

Dominio seleccionado: Sistema de gestión para veterinarias - Mascotas y Microchips

Justificación:

- **Relevancia práctica**: Las veterinarias requieren sistemas eficientes para gestionar información de mascotas y sus microchips
- **Complejidad adecuada**: La relación 1:1 entre Mascota y Microchip permite implementar conceptos avanzados sin excesiva complejidad
- **Aplicación real**: Los microchips son obligatorios en muchas jurisdicciones para identificación animal
- **Integridad de datos**: Permite demostrar el manejo de transacciones y relaciones constrained

Diseño de Modelado

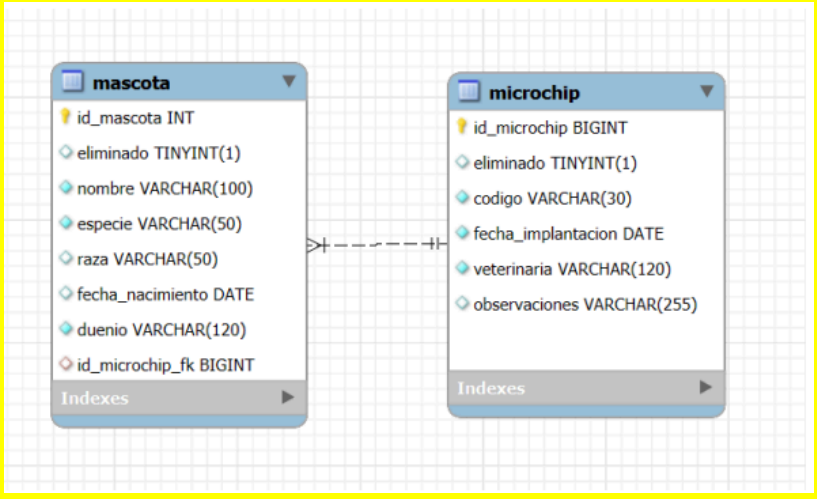
1. Relación 1→1 Unidireccional

- Mascota contiene referencia a Microchip
- No viceversa para simplificar el modelo
- Cohesión con el dominio real: cada mascota tiene un único microchip

2. Clave Foránea Única vs PK Compartida

- **Decisión**: FK única en tabla Mascota referenciando Microchip
- **Ventajas**:
    - Mantiene identidades separadas
    - Permite microchips sin mascota temporalmente
    - Flexibilidad para cambios futuros
- **Implementación**: mascota.microchip_id UNIQUE FOREIGN KEY

Diagrama UML (mascota - microchip)



Arquitectura por Capas: Estructura de Paquetes y Responsabilidades

/config

├── DatabaseConnection.java // Pool de conexiones y configuración

└── TransactionManager.java // Gestión transaccional

/model

├── Mascota.java // Entidad de dominio Mascota

└── Microchip.java // Entidad de dominio Microchip

/dao

├── GenericDAO.java // Interfaz con operaciones CRUD base

├── MascotaDAO.java // Interfaz específica Mascota

├── MascotaDAOImpl.java // Implementación JDBC para Mascota

├── MicrochipDAO.java // Interfaz específica Microchip

```
└── MicrochipDAOImpl.java     // Implementación JDBC para Microchip

/service
├── GenericService.java       // Interfaz servicio genérico
├── MascotaService.java       // Interfaz servicio Mascota
├── MascotaServiceImpl.java   // Lógica de negocio Mascota
├── MicrochipService.java     // Interfaz servicio Microchip
└── MicrochipServiceImpl.java // Lógica de negocio Microchip

/main
├── Main.java                 // Punto de entrada
├── AppMenu.java              // Navegación principal
├── MenuDisplay.java          // Visualización de menús
├── MenuHandler.java          // Procesamiento de opciones
└── TestConexion.java         // Pruebas de conectividad

/resources
└── db.properties             // Configuración base de datos
```

Estructura de Base de Datos

```
Sql
CREATE TABLE microchip (
   id_microchip BIGINT PRIMARY KEY AUTO_INCREMENT,
   eliminado TINYINT(1) DEFAULT 0,
   codigo VARCHAR(30) NOT NULL,
   fecha_implantacion DATE NOT NULL,
```

```
    veterinaria VARCHAR(120) NOT NULL,
    observaciones VARCHAR(255),

    CONSTRAINT UQ_Codigo UNIQUE (codigo),
    CONSTRAINT CHK_Fecha_Minima_Chip CHECK (fecha_implantacion >= '2000-01-01')
);

CREATE TABLE mascota (
    id_mascota INT AUTO_INCREMENT PRIMARY KEY,
    eliminado TINYINT(1) DEFAULT 0,
    nombre VARCHAR(100) NOT NULL,
    especie VARCHAR(50) NOT NULL,
    raza VARCHAR(50),
    fecha_nacimiento DATE,
    duenio VARCHAR(120) NOT NULL,

    CONSTRAINT chk_especie_valida CHECK (especie IN ('Perro', 'Gato', 'Ave', 'Roedor')),

    -- CONSTRAINT CHK_Fecha_Nacimiento CHECK (fecha_nacimiento < CURDATE()),

    id_microchip_fk BIGINT,

    CONSTRAINT UQ_Mascota__Microchip UNIQUE (id_microchip_fk),

    CONSTRAINT FK_Mascota_Microchip FOREIGN KEY (id_microchip_fk)
        REFERENCES microchip(id_microchip)
        ON UPDATE RESTRICT
        ON DELETE SET NULL
);
```

Orden de Operaciones y Transacciones: Flujo transaccional típico:

Java

```
// En MascotaServiceImpl.crearMascotaConMicrochip()
transactionManager.beginTransaction();
try {

   // 1. Persistir Microchip primero
   Microchip microchipCreado = microchipDAO.crear(microchip);

   // 2. Asignar microchip a mascota
   mascota.setMicrochipId(microchipCreado.getId());

   // 3. Persistir Mascota
   Mascota mascotaCreada = mascotaDAO.crear(mascota);

   // 4. Confirmar transacción
   transactionManager.commit();
   return mascotaCreada;

} catch (SQLException e) {

   // 5. Rollback en caso de error
   transactionManager.rollback();
   throw new ServiceException("Error en transacción", e);
}
```

Puntos de commit/rollback:

- **Commit**: Después de todas las operaciones DAO exitosas
- **Rollback**: En bloques catch ante cualquier excepción SQL
- **Auto-commit**: Deshabilitado durante transacciones

Validaciones de Entrada

Mascota:

- Nombre no vacío y longitud máxima 100 caracteres
- Especie dentro de valores permitidos: PERRO, GATO, AVE, ROEDOR, OTRO
- Fecha de nacimiento no futura

Microchip:

- Código único en el sistema
- Formato de código validado (alfanumérico)
- Fecha de implantación no futura
- Fabricante no vacío

Reglas de Negocio Implementadas

1. **Integridad relación 1:1**: No se permiten microchips duplicados en mascotas
2. **Baja lógica**: Las eliminaciones marcan eliminado = TRUE en lugar de DELETE físico
3. **Consistencia temporal**: Fechas no pueden ser futuras
4. **Unicidad**: Código de microchip debe ser único en el sistema
5. **Transaccionalidad**: Operaciones compuestas son atómicas

Pruebas Realizadas

Pruebas EN MYsql

Consultas SQL de Verificación: Consulta de mascotas con microchip:

Sql

```
SELECT m.nombre, m.espece, mc.codigo, mc.fecha_implantacion
FROM Mascota m
JOIN Microchip mc ON m.microchip_id = mc.id
WHERE m.eliminado = FALSE;
```

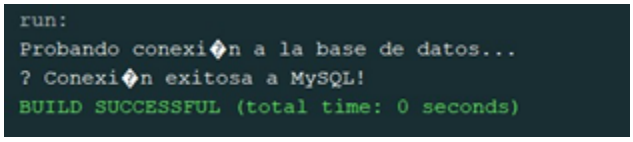
Verificación de integridad referencial:

Sql

```
SELECT COUNT(*) as mascotas_sin_microchip
FROM Mascota
WHERE microchip_id IS NOT NULL
AND microchip_id NOT IN (SELECT id FROM Microchip);
```

Pruebas en java con Capturas de Funcionamiento

(Ejecutar desde TestConextion.java → consola de NetBeans) para probar la conexión



(Ejecutar desde Main.java → consola de NetBeans)

PRUEBAS COMPLEJAS en Gestionar Mascotas

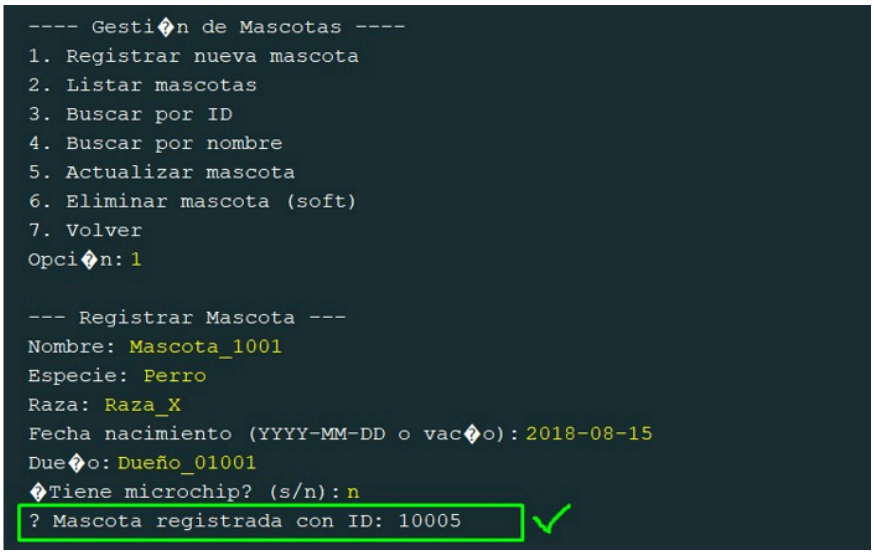
CASO 1 — CREAR MASCOTA (sin chip) — usar ID a partir de 1001

Acción en menú: 1 → Gestionar Mascotas → 1 Registrar nueva mascota

Entradas:

- Nombre: Mascota_1001
- Especie: Perro
- Raza: Raza_X
- Fecha nacimiento (YYYY-MM-DD o vacío): 2018-08-15
- Dueño: Dueño_01001
- ¿Tiene microchip? (s/n): n

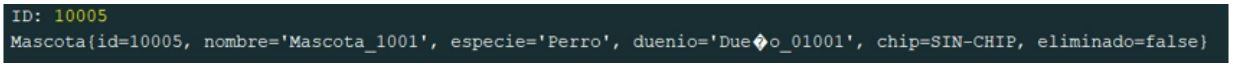
Resultado esperado en pantalla:



Qué hace (SQL):

INSERT INTO mascota (nombre, especie, raza, fecha_nacimiento, duenio, id_microchip_fk) VALUES ('Mascota_1001', 'Perro', 'Raza_X', '2018-08-15', 'Dueño_01001', NULL); Transacción: MascotaService.insertar() abre transacción y hace COMMIT al finalizar.

Verificación:



CASO 2 — CREAR MICROCHIP INDEPENDIENTE (ID >= 1001)

Acción en menú: 2 → Gestionar Microchips → 1 Registrar microchip

Entradas:

Código: MC-10001

Fecha implantación (YYYY-MM-DD): 2023-12-01

Veterinaria: Vet_David

Observaciones: Implante 1001

Resultado esperado:

```
---- Gesti□n de Microchips ----
1. Registrar microchip
2. Listar microchips
3. Buscar por ID
4. Buscar por c□digo
5. Actualizar microchip
6. Eliminar microchip (soft)
7. Volver
Opci□n: 1

--- Registrar Microchip ---
C□digo: MC-10001
Fecha implantaci□n (YYYY-MM-DD): 2023-12-01
Veterinaria: Vet_David
Observaciones (opcional): Implante 1001
? Microchip registrado con ID: 10008
```

SQL ejecutado:

INSERT INTO microchip (codigo, fecha_implantacion, veterinaria, observaciones) VALUES ('MC-10001', '2023-12-01', 'Vet_David', 'Implante 1001');

Transacción: MicrochipService.insertar() → COMMIT.

Verificación:

```
C□digo: MC-10001
Microchip{id=10008, codigo='MC-10001', fechaImplantacion=2023-12-01, veterinaria='Vet_David', eliminado=false}
```

CASO 3 — CREAR MICROCHIP Y ASOCIARLO A MASCOTA EXISTENTE (transacción)

Usar mascota ya existente, p.ej. mascota_ID = 10005 (del Caso 1)

Acción en menú: 2 → Registrar microchip → 7 Asociar microchip

Entradas sugeridas:

ID de mascota: 10005 (CASO 1)

ID de Mascota a asociar: MC-10001 (CASO 2)

Resultado esperado:

```
===== SISTEMA DE VETERINARIA  Mascotas & Microchips =====
1. Gestionar Mascotas
2. Gestionar Microchips
3. Salir
Opci n: 1

---- Gesti n de Mascotas ----
1. Registrar nueva mascota
2. Listar mascotas
3. Buscar por ID
4. Buscar por nombre
5. Actualizar mascota (NO asocia)
6. Eliminar mascota (soft)
7. Asociar / Desasociar Microchip
8. Volver
Opci n: 7
ID de mascota: 10005

Mascota encontrada:
Nombre: Mascota_1001
Especie: Perro
Raza: Raza_X
Fecha nacimiento: 2018-08-15
Due o: Due o_01001
Microchip asignado: (NINGUNO)

Seleccione una opci n:
1) Asignar microchip
0) Volver atr s
Opci n: 1
C digo del microchip: MC-10001
? Microchip asignado:
Mascota: Mascota_1001 ? Chip: MC-10001
```

Explicación:

1. 1 → Gestionar Mascotas

2. 7 → Asociar / Desasociar Microchip 3. El sistema solicita el ID de la mascota: ID de mascota: 10005

4. La consola muestra los datos de la mascota:

Mascota encontrada:

…

Microchip asignado: (NINGUNO)

5. El menú solicita acción:

1. Asignar microchip

2. Volver atrás

Opción: 1

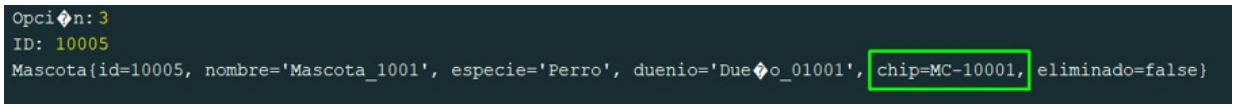
6. **El sistema solicita código del microchip:** Código del microchip: MC-10001

7. **Si el microchip existe y no está asociado, el sistema actualiza la mascota.**

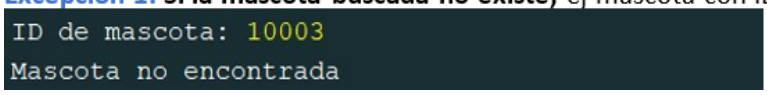
En DAO;

UPDATE mascota SET nombre=?, especie=?, raza=?, fecha_nacimiento=?, duenio=?, id_microchip_fk=? " WHERE id_mascota=?

Verificacion:



Excepción 1: Si la mascota buscada no existe; ej mascota con ID; 10003 (que no existe)



Descripción: El usuario ingresa un ID inexistente.

Ingreso:

ID de mascota: 10003 Respuesta esperada:

Mascota no encontrada

No se realizan cambios en base de datos.

Excepción 2: Microchip buscado no existe; ej microchip con Codigo:MC-9999 (que no existe)

```
Seleccione una opci□n:
1) Asignar microchip
0) Volver atr□s
Opci□n: 1
C□digo del microchip: MC-9999
? El microchip no existe. Debe registrarlo primero.
```

Descripción: La mascota existe, pero el código ingresado no corresponde a ningún microchip registrado. Ingreso:

Código del microchip: MC-9999 Respuesta esperada:

El microchip no existe. Debe registrarlo primero.

No se hace ningún UPDATE en mascota.

No hay commit sobre asociación.

ROLLBACK: Microchip buscado existe pero ya esta asignado;

```
Seleccione una opci□n:
1) Asignar microchip
0) Volver atr□s
Opci□n: 1
C□digo del microchip: MC-00001
? Error: Duplicate entry '910' for key 'mascota.UQ_Mascota__
Microchip'
```

Descripción: La mascota existe, pero el código ingresado ya esta siendo usado, como no hay validación en handler, service crea la coneccion y deja espuesto a dao, dao no puede ingresar ya que existe una Violación de UNIQUE (duplicar código de microchip)

-- (Asumir que ya existe un microchip con codigo 'MC-00001' de las cargas previas)

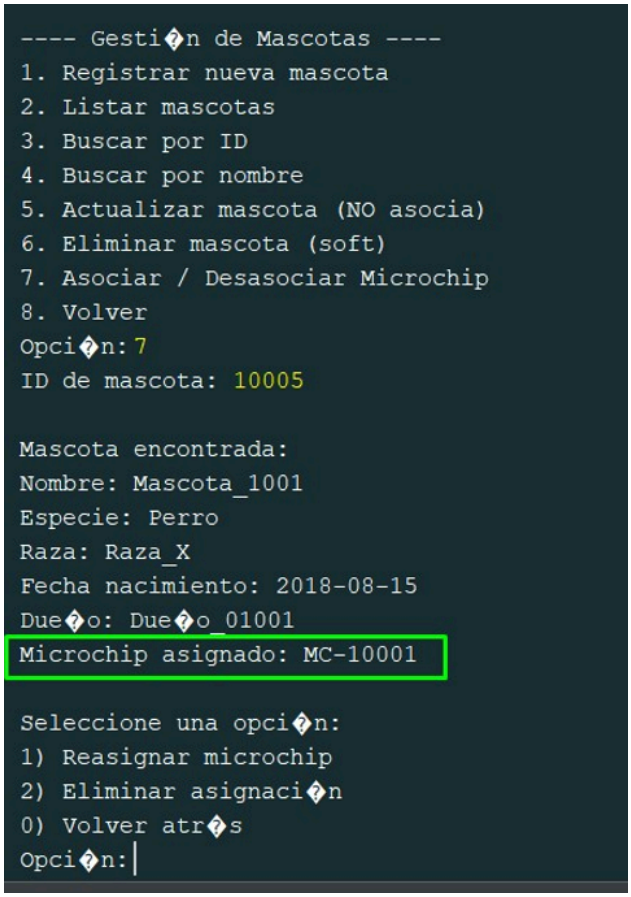
Verificación:

```
Opci□n: 3
ID: 1
Mascota{id=1, nombre='Mascota_910', especie='Gato', duenio='
Due□o_00910', chip=MC-00001, eliminado=false}
```

CASO 4 — INTENTAR ASOCIAR UN SEGUNDO MICROCHIP A LA MISMA MASCOTA

Escenario de prueba: mascota 10005 (CASO 1) ya tiene id_microchip_fk no NULL.

Acción en menú: 7 → Asociar microchip Entradas:



Explicación del Flujo: Asociar / Desasociar Microchip (CON UN MICROCHIP YA ASOCIADO

1. El usuario elige la opción “Asociar / Desasociar Microchip”

2. Se solicita el ID de la mascota y se busca en la base con mascotaService.getById(id). Si existe, se muestran sus datos.

Si no existe, salida posible: “Mascota no encontrada” CASO 3 excepción 1

3. Se muestra si la mascota ya tiene microchip asignado (como en la imagen adjutada “Microchip asignado: MC-10001”).

4. Aparece el submenú:

- **Opción 1:** Reasignar microchip Se pide el nuevo código. Se busca en BD:
  - *Si existe → se ejecuta actualización en mascota. Commit si la asignación es correcta **CASO 3**; rollback si falla.*

    - *Si no existe → salida: "El microchip no existe. Debe registrarlo primero".* ***CASO 3*** *excepción 2*
- **Opción 2:** Eliminar asignación Se ejecuta mascotaService.desasociarChip(idMascota). Aquí hay escritura en BD.
    - *Si se elimina correctamente → commit y mensaje: "Asignación eliminada"* ***CASO 6***
    - *Si ocurre un error → rollback o Opción 3: Volver al menú anterior No hay cambios en BD.*

CASO 5 — CREAR MASCOTA CON MICROCHIP EN UNA SOLA OPERACIÓN (insertar B luego A en la misma transacción)

Acción en menú: 1 → Registrar nueva mascota

(Responder s a "¿Tiene microchip?" y aportar los datos del microchip).

Entradas de ejemplo:

Nombre: Mascota_1101

Especie: Gato

Raza: Raza_Y

Fecha nac: 2020-10-10

Dueño: Dueño_01101

Tiene microchip? s

Código: MC-11001

Fecha implantación: 2022-02-02

Veterinaria: Vet_David

Observaciones: alta conjunta 1101

Resultado esperado:

```
---- Gesti□n de Mascotas ----
1. Registrar nueva mascota
2. Listar mascotas
3. Buscar por ID
4. Buscar por nombre
5. Actualizar mascota (NO asocia)
6. Eliminar mascota (soft)
7. Asociar / Desasociar Microchip
8. Volver
Opci□n: 1

--- Registrar Mascota ---
Nombre: Mascota_1101
Especie: Gato
Raza: Raza_Y
Fecha nacimiento (YYYY-MM-DD o vac□o): 2020-10-10
Due□o: Dueño_01101
□Tiene microchip? (s/n): s
C□digo del chip: MC-11001
? El microchip no existe. Debe registrarlo ahora.

--- Registrar Microchip ---
C□digo: MC-11001
Fecha implantaci□n (YYYY-MM-DD): 2022-02-02
Veterinaria: Vet_David
Observaciones (opcional): Implante 1009
? Microchip registrado con ID: 10012
? Microchip registrado con ID: 10012
? Mascota registrada con ID: 10007
```

Verificación:

```
Opci□n: 4
Nombre (LIKE): Mascota_1101
Mascota{id=1810, nombre='Mascota_1101', especie='Ave', duenio='Due□o_01101', chip=MC-01810, eliminado=false}
Mascota{id=10007, nombre='Mascota_1101', especie='Gato', duenio='Due□o_01101', chip=MC-11001, eliminado=false}
```

Explicación del Flujo en Consola (con puntos donde ocurre commit o rollback)

1. El usuario ingresa a la opción “Registrar nueva mascota”.

2. Se piden los datos básicos de la mascota (nombre, especie, raza, fecha de nacimiento, dueño). *Aún no hay operaciones en la base de datos.*

3. El sistema pregunta si tiene microchip. El usuario responde “s”. S*e solicita al usuario ingresar el código del chip.*

4. Se intenta buscar en BD el microchip mediante microchipService.getByCodigo(codigo). *Si el microchip no existe, no hay commit ni rollback: solo se muestra el mensaje correspondiente.*

5. Como el chip no existe, se ejecuta el formulario para registrarlo. Aquí se llama a microchipService.insertar(microchipNuevo).

- *En este momento se ejecuta la transacción de creación del microchip.*
- *Si la inserción es correcta, se produce un commit.*

- *Si ocurre algún error al insertar (por ejemplo, código duplicado), se realizaría un rollback.*

6. Una vez insertado, se vuelve a buscar el microchip con getByCodigo. *Esto es solo una consulta; no genera commit ni rollback.*

7. Luego se registra la mascota llamando a mascotaService.insertar(mascota).

- *Acá se ejecuta la transacción para crear la mascota. o Si la inserción es correcta, se realiza un commit.*
- *Si ocurre un error (por ejemplo, ID dueño inválido), corresponde un rollback.*

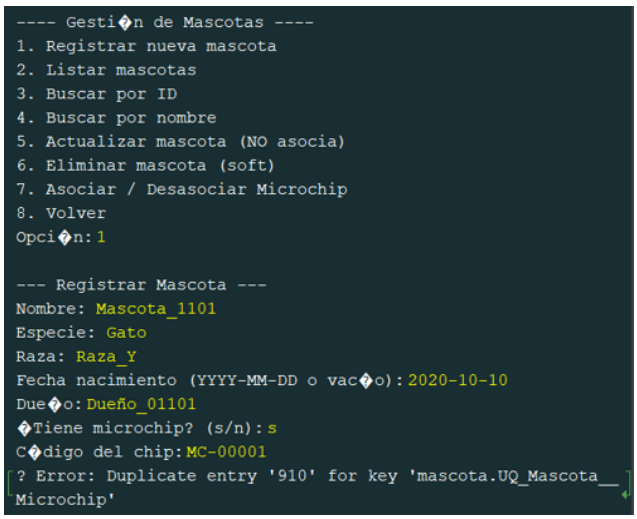
8. Finalmente se muestra por consola que el microchip fue registrado (con su ID) y que la mascota también fue registrada (con su ID).

Resumen de puntos con transacciones

Transacciones con commit/rollback:

- Insertar Microchip
- Insertar Mascota Consultas sin transacción:
- Buscar microchip por código
- Lectura de datos ingresados por el usuario

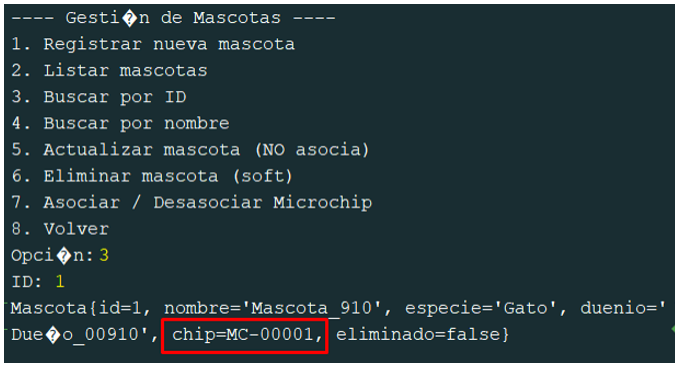
ROLLBACK: Microchip buscado existe pero ya está asignado;



Descripción: La mascota existe, pero el código ingresado ya esta siendo usado, como no hay validación en handler, service crea la coneccion y deja espuesto a dao, dao no puede ingresar ya que existe una Violación de UNIQUE (duplicar código de microchip)

-- (Asumir que ya existe un microchip con codigo 'MC-00001' de las cargas previas)

Verificación:



Explicación del Flujo en Consola (con puntos donde ocurre commit o rollback)

9. Se intenta buscar en BD el microchip mediante microchipService.getByCodigo(codigo). Si el microchip no existe, no hay commit ni rollback: solo se muestra el mensaje correspondiente.

10. Si ocurre algún error al insertar (por ejemplo, código duplicado), se realizaría un **rollback.**

BREVE EXPLICACIÓN: ¿POR QUÉ COMMIT O ROLLBACK EN CADA CASO?

- **COMMIT** → se usa cuando todas las operaciones dependientes (insert microchip, update mascota, etc.) se completan sin excepción. Guarda cambios en forma permanente.

- **ROLLBACK** → se usa cuando ocurre una excepción (violación UNIQUE, CHECK, FK, error lógico o fallo forzado). Revierte todo lo hecho en esa transacción para mantener la base consistente.

- **Validación previa (en Service)** evita excepciones lanzadas por la BD y permite dar mensajes amigables (p. ej. “esta mascota ya tiene microchip”). Si se detecta la condición, el Service evita ejecutar el INSERT/UPDATE y salta rollback/exception.

Conclusiones y Mejoras Futuras

Conclusiones

1. **Arquitectura efectiva**: La separación en capas demostró ser mantenible y testeable
2. **Transacciones necesarias**: La gestión transaccional fue crucial para la integridad de la relación 1:1
3. **JDBC adecuado**: El uso directo de JDBC permitió control granular sobre las operaciones
4. **Baja lógica acertada**: Preservar datos históricos resultó valioso para el dominio

Mejoras Futuras

1. **Frontend web**: Migrar de consola a interfaz web con Spring Boot
2. **API REST**: Exponer servicios como API RESTful
3. **Logging**: Implementar sistema de logging robusto
4. **Caché**: Agregar caché de segundo nivel para mejor performance
5. **Reportes**: Generación de reportes PDF de historiales médicos
6. **Auditoría**: Tracking de cambios en entidades críticas

Fuentes y Herramientas Utilizadas

Tecnologías

- **Java 21**: Lenguaje de programación principal
- **MySQL 8**: Sistema de gestión de base de datos
- **JDBC**: API de conexión a base de datos
- **XAMPP**: Entorno de desarrollo con MySQL

Herramientas de Desarrollo

- **NetBeans 17/18**: IDE principal
- **Git**: Control de versiones
- **MySQL Workbench**: Diseño y administración de base de datos

Referencias Bibliográficas

1. Oracle Java Documentation - JDBC Guide
2. MySQL 8.0 Reference Manual