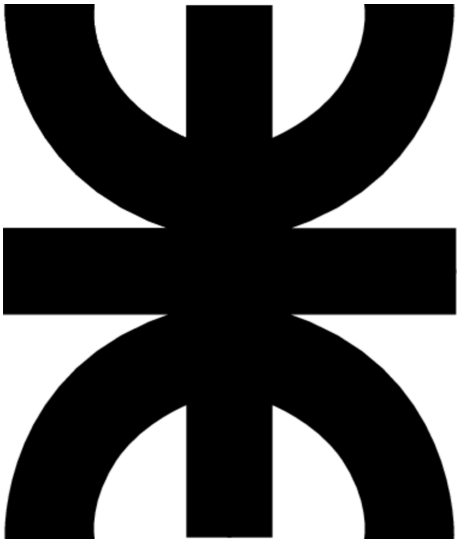
Trabajo Final Integrador (TFI)

UNIVERSIDAD TECNOLÓGICA NACIONAL



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

Dominio

- DispositivoIoT → ConfiguracionRed

Integrantes

- Ruidiaz, Emanuel Facundo – emafruidiaz@gmail.com
- Cristian Serna – sernachristian700@gmail.com
- Juan Martín Roques Zeballos – juanmartinroqueszeballos@gmail.com
- Mauro Gonzalo Santini – mgs.argentum@gmail.com

1. Introducción y Equipo

1.1 Integrantes y Roles

El proyecto fue desarrollado por el siguiente equipo:

- Ruidiaz, Emanuel Facundo: Desarrollador 1
- Cristian Serna: Desarrollador 2
- Juan Martín Roques Zeballos: Desarrollador 3
- Mauro Gonzalo Santini: Desarrollador 4

1.2 Elección y Justificación del Dominio

El dominio seleccionado es **DispositivoIoT (A) → ConfiguracionRed (B)**.

- **Justificación**: Este modelo representa claramente un escenario donde la existencia de una ConfiguracionRed depende directamente y de forma exclusiva de un único DispositivoIoT. Un sensor sólo puede tener una configuración de red activa (IP, Máscara) a la vez, garantizando la relación 1:1. Es un caso práctico de persistencia de datos relacionados.

2. Diseño de la Base de Datos y UML

2.1 Decisión Clave: Relación 1:1

Para garantizar la relación 1:1 en la base de datos relacional (MySQL), se eligió la técnica de Clave Foránea Única en la tabla dependiente (B).

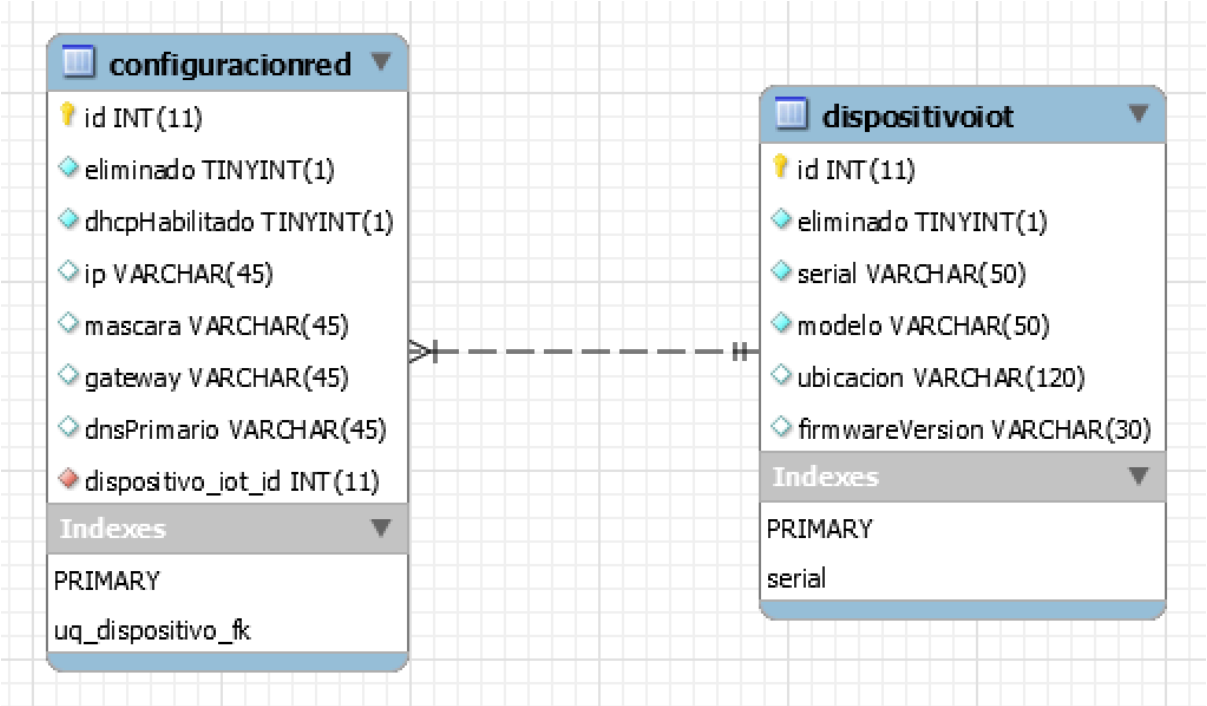
- Tabla B (ConfiguracionRed) contiene el campo dispositivo_iot_id.
- Este campo dispositivo_iot_id está definido como FOREIGN KEY a DispositivoIoT(id) y, crucialmente, tiene una restricción UNIQUE.
- La restricción UNIQUE asegura que un ID de dispositivo (A) no pueda aparecer más de una vez en la tabla de configuraciones (B).
- Baja Lógica: Ambas tablas incluyen el campo eliminado BOOLEAN NOT NULL DEFAULT 0 para ocultar registros sin borrarlos físicamente.

2.2 Esquema DDL de la Relación

El script SQL que garantiza la integridad de la relación es el siguiente

```
CREATE TABLE DispositivoIoT (
    id INT PRIMARY KEY AUTO_INCREMENT,
    eliminado BOOLEAN NOT NULL DEFAULT 0,
    serial VARCHAR(50) NOT NULL UNIQUE,
    modelo VARCHAR(50) NOT NULL,
    ubicacion VARCHAR(120) NULL,
    firmwareVersion VARCHAR(30) NULL,
    CHECK (eliminado IN (0, 1))
);
```

2.3 Diagrama UML de Clases



Scripts de creación y carga de datos adjuntos en el repositorio.

- DDL_Creacion_Tablas.sql
- DML_Datos_Prueba.sql

3. Arquitectura del Proyecto y Responsabilidades

El proyecto sigue una arquitectura de tres capas bien definida, utilizando paquetes para separar las responsabilidades:

Capa	Paquete	Responsabilidad Principal
Presentación	main/	Interacción con el usuario (AppMenu). Recolección de datos y muestra de resultados. Solo llama a la Capa Service.
Service	service/	Lógica de Negocio, Validaciones y Control Transaccional. Orquesta las operaciones entre DAOs, iniciando commit o rollback.
Persistencia	dao/	Acceso y manipulación directa de la base de datos (CRUD). Ejecuta

		PreparedStatement y mapea ResultSet a entidades.
Modelo	entities/	Estructura de datos (DTO). Contenedor simple sin lógica de negocio.
Configuración	config/	Manejo de credenciales (database.properties) y utilidades de conexión (DatabaseConnection, TransactionManager).

4. Persistencia y Gestión Transaccional

4.1 Orden de Operaciones para la relación 1:1

Para la operación de Creación compuesta (crearDispositivoConConfiguracion), la secuencia transaccional es crítica:

1. Crear A (DispositivoIoT): Se inserta el dispositivo para que la base de datos le asigne su ID (PK).
2. Obtener ID: El DispositivoIoTDAOImpl recupera el id generado (Statement.RETURN_GENERATED_KEYS
3. Asignar FK: La Capa Service toma el id de A y lo asigna como dispositivoIotId (la FK) en la entidad B.
4. Crear B (ConfiguracionRed): Se inserta la configuración, utilizando el id de A para satisfacer la restricción UNIQUE.

4.2 Control de Transacciones

El control atómico de las operaciones (A y B) reside en la capa Service (DispositivoIoTServiceImpl.java), encapsulado por el TransactionManager.

```
// DispositivoIoTServiceImpl.java
 try {
 conn = tx. beginTransaction (); // 1. setAutoCommit(false)

 // 2. Crear A (obtiene ID)
 dispDAO.crearTx(dispositivo , conn);

 // 3. Asignar FK
 dispositivo . getConfiguracionRed (). setDispositivoIotId ( dispositivo .getId ());

 // 4. Crear B
 confDAO.crearTx( dispositivo . getConfiguracionRed (), conn);
```

```
tx.commit (); // 5. Confirmar si ambas operaciones fueron exitosas
} catch ( Exception e) {
tx.rollback (); // 6. Deshacer si B Falla o hay cualquier error
throw e;
}
```

5. Validaciones y Reglas de Negocio

5.1 Tipos de Validaciones Implementadas

- **Validación de Existencia y No Nulos (Service):** Se verifica que campos cruciales como serial y la referencia configuracionRed no sean nulos antes de iniciar la transacción.
- **Validación de Unicidad (Service):** Antes de crear un dispositivo, el Service llama al DAO (dispDAO.findBySerial()) para verificar si el serial ya está en uso, evitando violaciones de la restricción UNIQUE en la base de datos.
- **Baja Lógica (DAO)**: La regla de negocio dicta que el método eliminar() del Service se mapea a updateEstadoEliminado(id, true) en el DAO, modificando el campo eliminado en lugar de ejecutar un DELETE físico.

6. Pruebas Realizadas

Se realizaron pruebas funcionales a través del menú de consola para verificar la correcta implementación del CRUD y la integridad transaccional.

6.1 Prueba Transaccional Exitosa (Commit)

- Acción: Opción 1 del menú (Crear nuevo Dispositivo).
- Resultado Esperado: El INSERT en DispositivoIoT y el INSERT en ConfiguracionRed se ejecutan correctamente, y tx.commit() hace permanentes los dos registros en la base de datos, manteniendo la FK única.

6.2 Prueba Transaccional Fallida (Rollback)

- Simulación de Error: Intentar insertar un DispositivoIoT con un serial que ya existe (serial UNIQUE violado).
- Resultado Observado: La Capa Service lanza una IllegalArgumentException (debido a la validación de unicidad previa). Si el error ocurriera entre el INSERT de A y el INSERT de B, tx.rollback() se ejecuta, asegurando que el registro de A (si se hubiera insertado) no persista, evitando datos huérfanos.

6.3 Consultas SQL Útiles de Verificación

```
-- 1. Verificar registros activos (eliminado = 0)
 SELECT * FROM DispositivoIoT WHERE eliminado = 0;

 -- 2. Verificar la carga de la relación 1:1 desde A
SELECT
d.serial ,
c.ip ,
c. dhcpHabilitado
FROM
DispositivoIoT d
JOIN
ConfiguracionRed c ON d.id = c. dispositivo_iot_id
WHERE
d. eliminado = 0;
```

7. Conclusiones y Mejoras Futuras

7.1 Conclusiones

El proyecto cumple satisfactoriamente con todos los requisitos del TFI. Se logró la separación efectiva de las preocupaciones utilizando el patrón DAO y la Capa Service. La gestión transaccional centralizada en el Service garantiza la atomicidad de las operaciones compuestas, siendo fundamental para mantener la integridad de la relación 1:1. El uso de PreparedStatement y try-with-resources asegura un código limpio, seguro contra la inyección SQL y eficiente en el manejo de recursos.

7.2 Mejoras Futuras

- **Genericidad de la Baja Lógica:** Actualmente, la baja lógica solo se aplica a la tabla A. En futuras mejoras, la operación de eliminarLogico debería ser una transacción que actualice el campo eliminado = 1 en ambas tablas (A y B) para mantener la consistencia de la baja lógica en toda la entidad compuesta.
- **Patrón Singleton para DAO/Service:** Usar un patrón Singleton o un gestor de **Contexto** para instanciar los DAOs y Services solo una vez y pasarlos, mejorando la eficiencia y limpieza del código en el Main.
- **Actualización Transaccional:** Implementar la operación updateTx para permitir la modificación atómica de los datos de A y B simultáneamente.

8. Fuentes y Herramientas Utilizadas

- IDE: Apache NetBeans / IntelliJ IDEA.
- Base de Datos: MySQL 8.0+ (XAMPP).
- Lenguaje: Java SE.
- Librería: MySQL Connector/J (JDBC).