Sistema de Gestión de Empresas y Domicilios Fiscales

Trabajo práctico Integrador - Programación 2

Alumna: Sotelo Sofía

Profesores: Cinthia Rigoninthi y Jerónimo Cortez

Comisión: 8

Índice

---

1. Introducción

El presente trabajo práctico integrador desarrolla un sistema de gestión que implementa el dominio **Empresa ➜ DomicilioFiscal**. Se eligió esta pareja de entidades por su claridad para representar la asociación unidireccional 1 a 1 que es el foco principal del TPI. Se utiliza Java, MySQL, JDBC, transacciones, y arquitectura en capas, conforme a la rúbrica de evaluación.

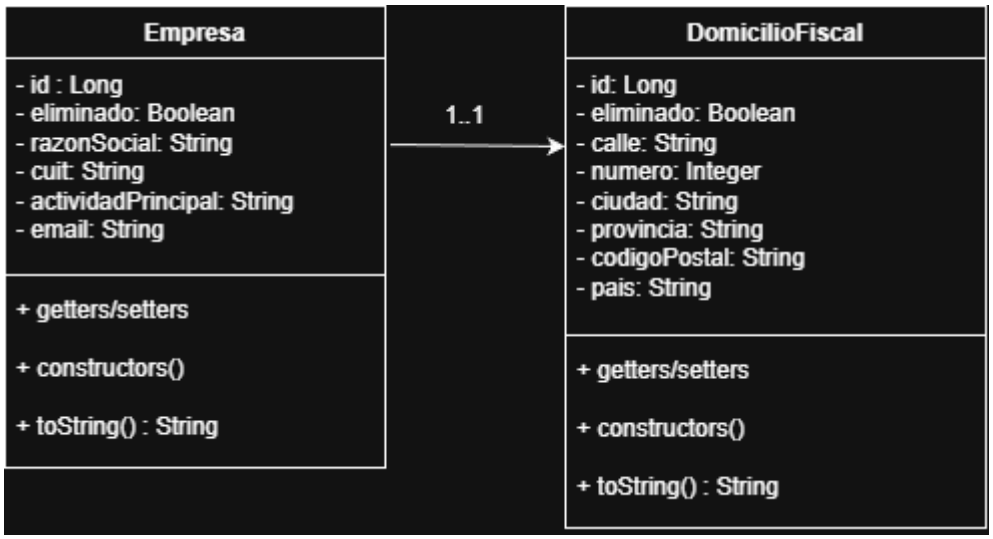
- **Empresa (Clase A):** Entidad principal, identifica a una persona jurídica.
- **DomicilioFiscal (Clase B):** Entidad dependiente, contiene la información necesaria para el registro fiscal de la empresa.

---

2. Diseño de Clases y Estructura

2.1. Diagrama UML de Clases

Se utilizó la clase abstracta BaseEntity para implementar la baja lógica (eliminado: Boolean) en ambas clases.



2.2. Decisiones Clave en la Relación 1→1

Se optó por la implementación más robusta de la relación 1-a-1, que es poner la **Clave Foránea (FK)** en la tabla dependiente **(domicilio_fiscal)** con una restricción de unicidad.

- **Tabla Empresa:** Contiene la clave primaria (id BIGINT).
- **Tabla DomicilioFiscal:** Contiene la columna empresa_id con las restricciones:
  - NOT NULL: Garantiza que el domicilio *siempre* esté ligado a una empresa.
  - UNIQUE KEY: Garantiza que *solo* una empresa pueda ligarse a ese domicilio.
  - Este diseño requirió incluir el campo empresaId: Long en la entidad Java DomicilioFiscal para poder pasar el ID a la capa DAO.

2.3. Arquitectura por Capas y Responsabilidades

Paquete	Capa Arquitectónica	Responsabilidad Principal
main	Presentación (UI)	Lector de entradas (Scanner) y llamada a Service.
service	Negocio	**Validaciones**, orquestación de Transacciones (commit/rollback).
dao	Persistencia	Ejecución de PreparedStatement y mapeo (uso exclusivo de SQL).
entities	Dominio	Estructura de datos (Empresa, DomicilioFiscal, BaseEntity).
config	Infraestructura	Gestión de la Conexión (DatabaseConnection) y transacciones (TransactionManager).

3. Persistencia y Transacciones

3.1. Gestión Transaccional (Lógica Atómica)

La funcionalidad más crítica es la inserción de una nueva Empresa con su DomicilioFiscal. Esta es una **operación atómica**.

- **Dónde se Controla:** La orquestación ocurre en el método EmpresaService.insertar(Empresa empresa).
- **Mecanismo:** Se utiliza la clase TransactionManager (implementando AutoCloseable).

```
@Override
public void insertar(Empresa empresa) throws Exception {
    validarEmpresa(empresa);

    try (Connection conn = DatabaseConnection.getConnection();
         TransactionManager tx = new TransactionManager(conn)) {

        tx.startTransaction();
```

3.2. Orden de Operaciones y Rollback

El orden de ejecución garantiza la integridad referencial y demuestra el rollback requerido:

1. **Inicio de Transacción:** tx.startTransaction() (pone autoCommit=false).
2. **Operación A (Empresa):** empresaDao.crear(...). Se obtiene el ID autogenerado de la empresa.
3. **Asociación:** Se asigna el ID obtenido al domicilio: domicilio.setEmpresaId(empresa.getId()).
4. **Operación B (Domicilio):** domicilioDao.crear(...). Se usa el ID recién generado para el INSERT.
5. **Cierre:** Si no hay errores, se llama a tx.commit(). Si hay un error (ej. **el fallo simulado**), el bloque try-with-resources llama automáticamente a TransactionManager.close(), que ejecuta el rollback(), dejando la BD intacta.

---

4. Reglas de Negocio y Validaciones

4.1. Validaciones en Capa Service

- **CUIT Único:** EmpresaService llama a empresaDao.leerPorCuit() para verificar que el CUIT no esté registrado antes de iniciar cualquier transacción.
- **Formatos:** Se realizan validaciones básicas de formato de CUIT (Regex) y email.
- **Campos Obligatorios:** Las validaciones (validarEmpresa, validarDomicilio) verifican los campos NOT NULL del esquema SQL.

4.2. Eliminación Segura (Soft Delete)

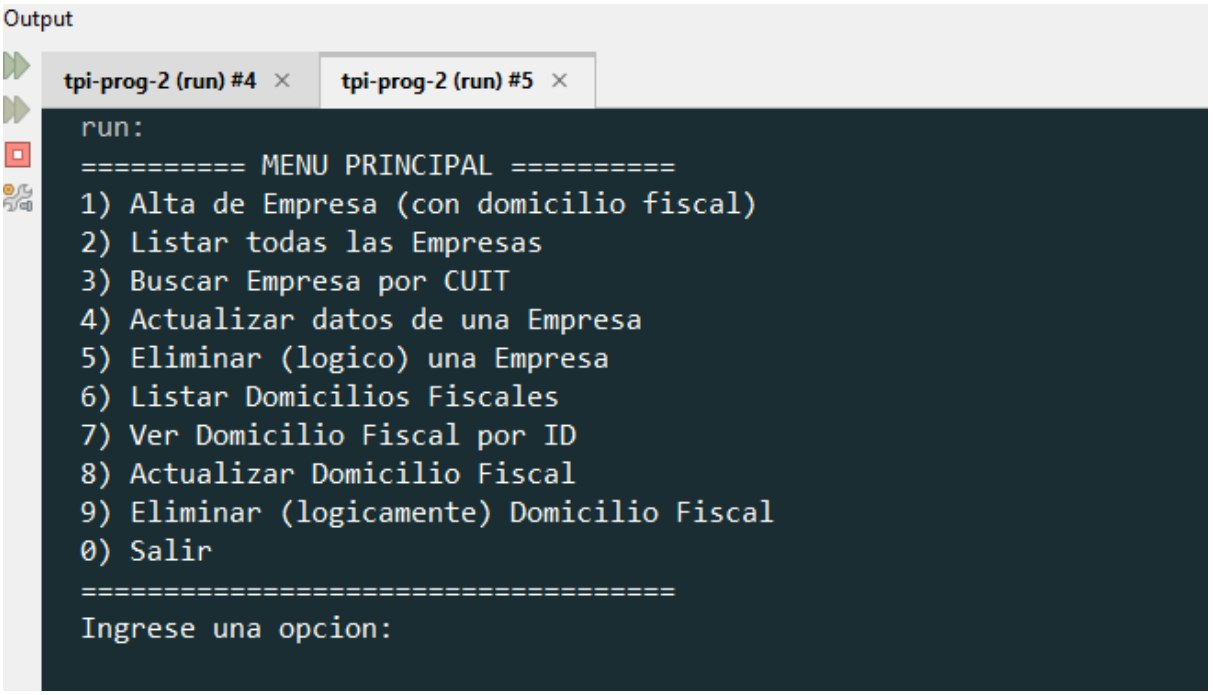
Se implementó la **Baja Lógica** (eliminado = TRUE). En el método EmpresaService.eliminar(id), se ejecuta una transacción para:

1. Obtener la empresa y su domicilio.
2. Eliminar lógicamente el DomicilioFiscal.
3. Eliminar lógicamente la Empresa.

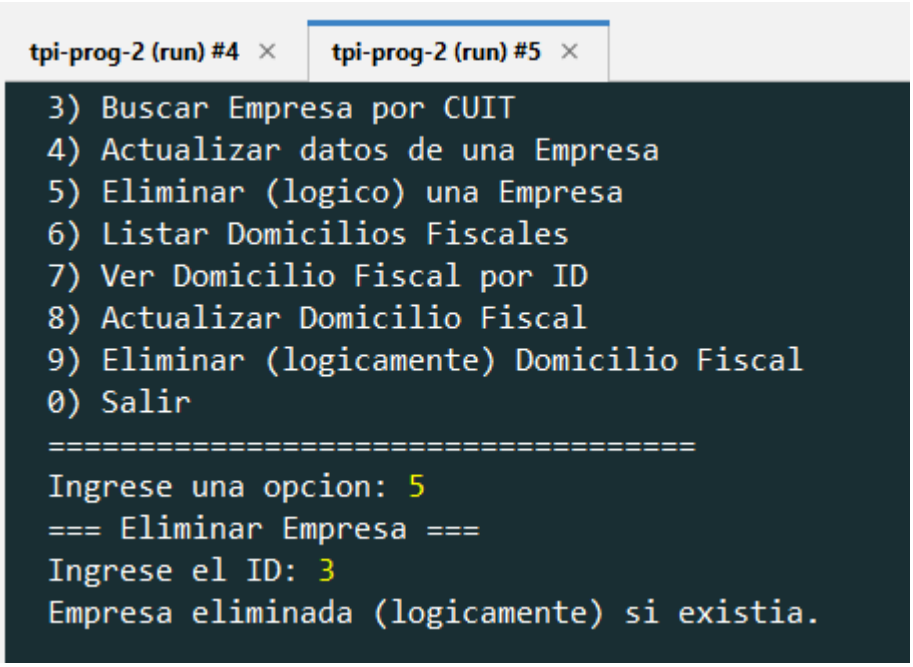
---

5. Pruebas realizadas

Menú

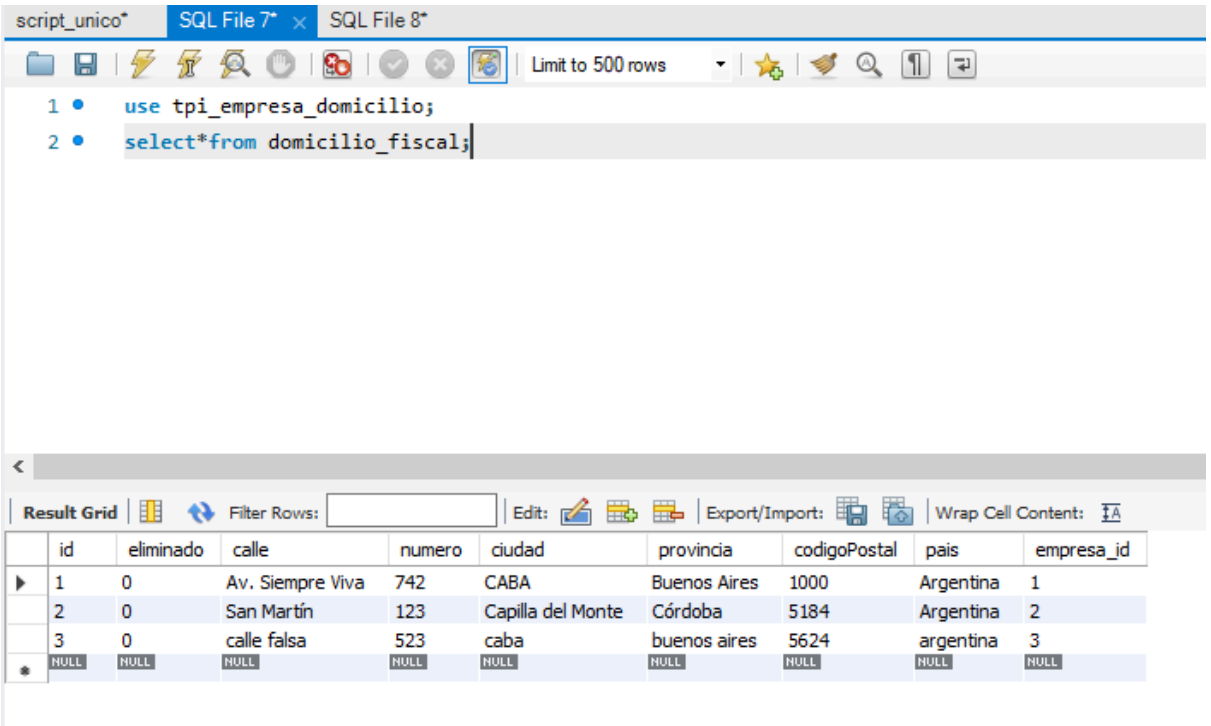


Baja lógica

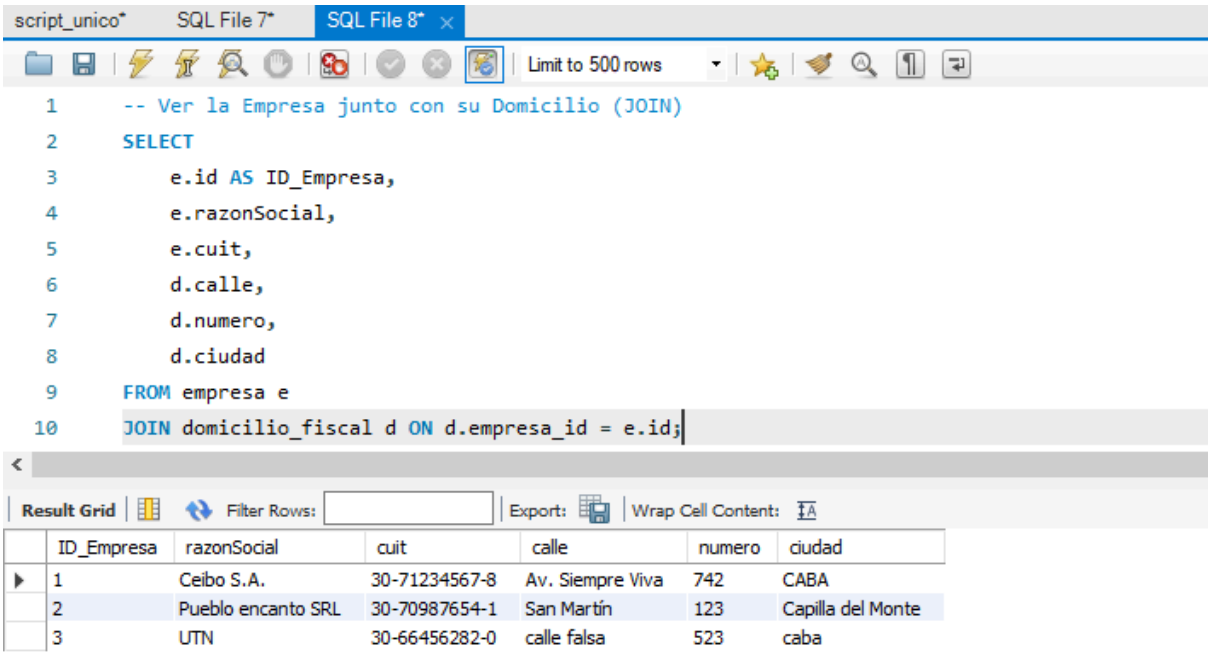


5.1. Consultas SQL Útiles

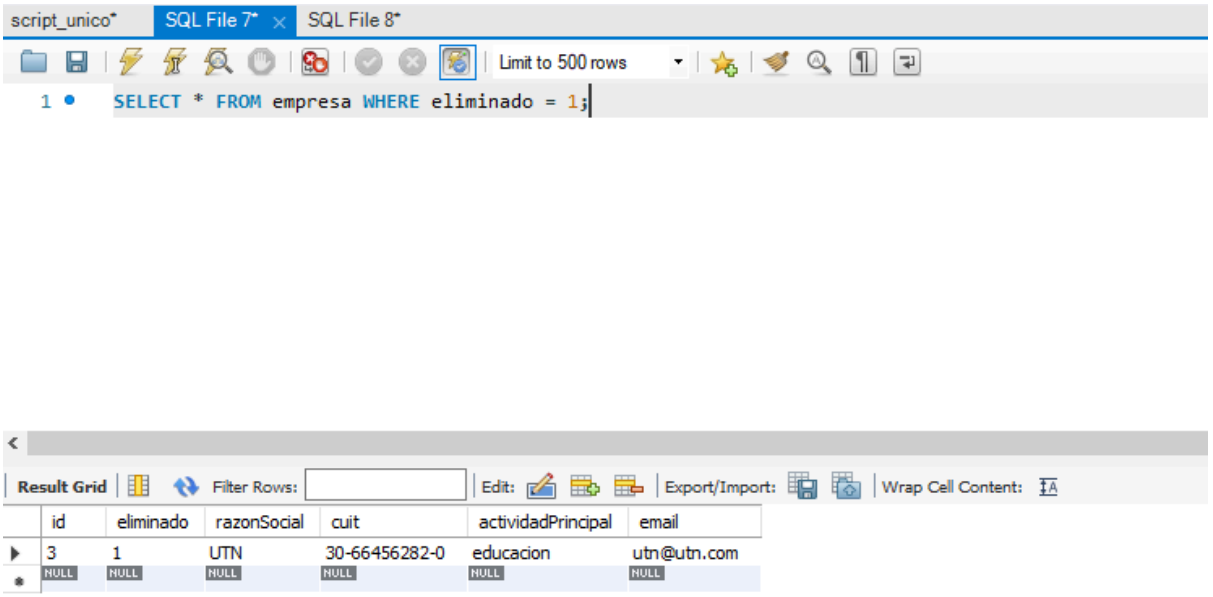
Consulta General



Consulta de unión



Consulta baja lógica



6. Conclusiones

El desarrollo de este Trabajo Final Integrador permitió consolidar los conceptos fundamentales de la Programación Orientada a Objetos y la persistencia de datos en Java.

A través de la implementación del dominio **Empresa - Domicilio Fiscal**, se alcanzaron las siguientes conclusiones técnicas y académicas:

- **Arquitectura y Desacoplamiento:** La adopción de una arquitectura en capas (**DAO, Service, Controller/Main**) demostró ser esencial para la mantenibilidad del software. Separar la lógica de negocio (Service) del acceso a datos (DAO) permite realizar cambios en las reglas de validación sin afectar las consultas SQL, y viceversa .
- **Integridad Transaccional:** El uso del patrón `TransactionManager` permitió garantizar la **atomicidad** en la creación de la Empresa y su Domicilio. Se comprobó experimentalmente cómo el mecanismo de `rollback` protege la base de datos ante fallos, evitando inconsistencias y registros "huérfanos" .
- **Robustez del Diseño 1-a-1:** Las restricciones estrictas en la base de datos (`NOT NULL`, `UNIQUE`) y validaciones previas en la capa de Servicio, demostraron la importancia de evitar operaciones inválidas.
- **Valor del JDBC Puro:** Trabajar con JDBC puro proporcionó un entendimiento profundo del ciclo de vida de las conexiones, el manejo de sentencias preparadas (`PreparedStatement`) para la seguridad y el mapeo manual de resultados, cumpliendo con los estándares de seguridad requeridos.

6.1. Desafíos y mejoras

El proyecto presentó el desafío adicional de la reestructuración del equipo, lo que conllevó a asumir la responsabilidad integral del desarrollo. Además tuve que darme de baja a la materia de base de datos, por lo cuál tuve que aprender cosas desde 0. Esto resultó en una oportunidad de aprendizaje intensivo, permitiendo una visión completa de cómo interactúan todas las piezas del sistema, desde el diseño UML hasta la ejecución de las consultas SQL. Como mejora, me gustaría incorporar validaciones de dominio más estrictas (CUIT con regex, email válido, ciudad/provincia/país no numéricos).

---

7. Bibliografía

- Oracle. (2024): https://docs.oracle.com/javase/tutorial/jdbc/
- MySQL. (2024). *MySQL 8.0 Reference Manual: COMMIT, ROLLBACK Statements*: https://dev.mysql.com/doc/refman/8.0/en/commit.html
- Alur, D., Crupi, J., & Malks, D. (2003). *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall. (Patrón Data Access Object).
- Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley. (Service Layer).
- https://youtu.be/mhDqL2SUXJc?si=kfL2UJYAyCmmF9mO
- IA