Trabajo Practico Integrador (TPI)

Programación 2 – Tecnicatura Universitaria en Programación

Universidad Tecnológica Nacional (UTN) – Modalidad a Distancia

Integrantes del Grupo 85

INTEGRANTES:

- Sandra Débora Martínez — Entidades, documentación DAO (presentación teórica).
- Melisa Inés Martellini — UML, modelado SQL, creación de tablas, claves, PK compartida, DatabaseConnection.
- Fabricio Nicolás Puccio — Capa Service, transacciones, validaciones, orquestación de reglas de negocio.
- Julián Daniel Gómez — DAOs completos, entidades finales, SQL, AppMenu, pruebas de integración, estructura general del proyecto

---

Introducción

El presente informe describe el desarrollo del Trabajo Final Integrador (TPI) de la materia *Programación 2*, correspondiente al Grupo 85. El objetivo fue diseñar e implementar un sistema CRUD completo utilizando **Java**, **JDBC** y **MySQL**, aplicando los principios fundamentales de la programación orientada a objetos, la arquitectura por capas y las buenas prácticas de persistencia de datos.

El dominio elegido fue un **Sistema de Gestión Bibliotecaria**, compuesto por las entidades **Libro** y **FichaBibliografica**, que representan tanto los datos físicos del libro como sus metadatos bibliográficos. Este dominio resulta especialmente adecuado para un proyecto académico, dado que permite aplicar conceptos teóricos como **relaciones 1→1**, **integridad referencial**, **claves primarias compartidas**, **validaciones de negocio**, **borrado lógico** y **transacciones atómicas**.

Durante el desarrollo se utilizaron patrones y técnicas estudiadas en la cursada, entre ellas:

- **Patrón DAO (Data Access Object)** para desacoplar la lógica de negocio de la persistencia.
- **Arquitectura por capas**, dividiendo el sistema en Entities → DAO → Service → Presentación.
- **Validaciones centralizadas** según reglas del dominio.
- **Manejo manual de transacciones** con commit() y rollback() para garantizar atomicidad.
- **PreparedStatement** para prevenir inyección SQL y mejorar rendimiento.
- **Modelo de PK compartida** para relaciones 1→1 estrictas.

El informe presenta las decisiones técnicas tomadas, el diseño implementado, las pruebas funcionales realizadas mediante menú por consola y las conclusiones finales del proyecto.

---

1. Dominio elegido y justificación

Se eligió como dominio un **sistema de gestión bibliotecaria** centrado en dos entidades principales:

- **Libro**
- **FichaBibliografica**

Estas entidades permiten modelar información esencial de una biblioteca real, incluyendo datos físicos del ejemplar (Libro) y metadatos bibliográficos (Ficha).
La decisión se justifica porque:

- Permite implementar de manera clara una relación **1→1** obligatoria o compartida por PK.
- Se adapta perfectamente a CRUD con validaciones reales: ISBN único, restricciones de longitud, año válido, etc.
- Permite incluir transacciones (crear libro + ficha juntas).
- Su complejidad es suficiente para cumplir los requerimientos del TPI sin generar un modelo excesivo.

2. Decisiones de diseño (1→1, PK compartida, FK, SQL)

Relación 1→1

Para implementar la relación 1→1, se evaluaron estas opciones:

1. **FK única** → libro.id → ficha.id
2. **PK compartida** (decisión final)

Razones de la decisión (PK compartida):

- Garantiza integridad *real*: cada libro tiene exactamente una ficha.
- Evita valores nulos y ambigüedades.
- Simplifica JOINs.
- Se integra bien con transacciones (crear libro → libro.id → usarlo como id de ficha).

DDL principal

- libro(id PK AUTO_INCREMENT, ...)
- ficha_bibliografica(id PK FK(libro.id), ISBN UNIQUE, ...)
- Ambas tablas manejan **eliminación lógica** con un booleano eliminado.

3. Arquitectura del proyecto

La aplicación se diseñó utilizando una arquitectura por capas, un enfoque recomendado en ingeniería de software para garantizar separación de responsabilidades, aislamiento de cambios y mantenibilidad del sistema. Este modelo sigue principios del diseño modular, como SRP (Single Responsibility Principle) y bajo acoplamiento, permitiendo que cada capa cumpla un rol bien **definido.**

La aplicación se implementó en **arquitectura por capas**, separando responsabilidades:

src/

├── entities/ → Modelo de dominio (Libro, FichaBibliografica)

├── dao/ → Acceso a datos (DAO + SQL + PreparedStatement)

├── service/ → Lógica de negocio, validaciones, transacciones

│ └── validations/ → Validación centralizada

├── config/ → DatabaseConnection y propiedades externas

├── main/ → AppMenu (interfaz de usuario por consola)

└── sql/ → Sentencias SQL organizadas por clase

4. Persistencia y transacciones

Conexión

Implementada mediante:

DatabaseConnection.getConnection()

- Permite cambiar credenciales sin tocar el código.
- Maneja errores (IOException + SQLException).
- Utiliza db.properties.

DAO

Cada DAO:

- Implementa CRUD mediante PreparedStatement.
- Usa SQL predefinido en clases constantes (LibroSQL, FichaBibliograficaSQL).
- Filtra automáticamente por eliminado = false.
- Tiene métodos con y sin Connection para transacciones.

Transacción clave: crear Libro + Ficha

conn.setAutoCommit(false);

libroDAO.crear(libro, conn);

ficha.setId(libro.getId());

fichaDAO.crear(ficha, conn);

conn.commit();

Si ocurre cualquier error → rollback().

Esta operación asegura atomicidad: o se crea ambas entidades o ninguna.

5. Validaciones y reglas de negocio

Validaciones principales

Libro

- Título obligatorio (≤150)
- Autor obligatorio (≤120)
- Año de edición válido
- Editorial opcional

Ficha

- ISBN obligatorio y único
- Formato: [0-9Xx-]+
- Límites de caracteres en Dewey, estantería e idioma

Reglas de negocio aplicadas

- **RN1:** ISBN único

- **RN2:** 1→1 mediante PK compartida
- **RN3:** Eliminación lógica
- **RN4:** Validaciones antes del DAO
- **RN5:** Commit / rollback en operaciones críticas
- **RN6:** Listados solo muestran registros no eliminados
- **RN7:** PreparedStatement para evitar SQL injection

6. Conclusiones y mejoras propuestas

Conclusiones

- Se logró un sistema completamente funcional con CRUD + validaciones + transacciones.
- La arquitectura por capas permitió dividir el trabajo entre integrantes.
- Se aplicó un diseño profesional: PK compartida, DAO limpio, PreparedStatement, menú usable.
- El código quedó modular y ampliable.

Mejoras futuras

- Migrar a **Spring Boot** (inyección de dependencias, repositorios).
- Agregar logs con **Log4j2**.
- Implementar **tests unitarios** JUnit + Mockito.
- Agregar soporte para otros materiales: revistas, películas, etc.
- Implementar interfaz gráfica (JavaFX).

8. Herramientas y fuentes

- Java 17
- JDBC
- MySQL 8
- IntelliJ IDEA
- ChatGPT (IA) para asistencia en redacción y refinamiento de código
- Documentación oficial de MySQL y JDBC

CONCLUSIÓN

El sistema desarrollado permitió aplicar de forma práctica los principios teóricos vistos en la materia, especialmente en lo referido a diseño modular, persistencia y relaciones entre entidades. Se comprobó que la arquitectura por capas facilita la escalabilidad y el mantenimiento, al distribuir responsabilidades de forma clara entre las clases. La implementación del patrón DAO demostró ser efectiva para mantener el desacoplamiento entre la lógica de negocio y la base de datos, favoreciendo la extensibilidad.

El uso de **transacciones JDBC** permitió garantizar la **consistencia del modelo 1→1**, especialmente en la creación conjunta de Libro y FichaBibliografica, donde la atomicidad es fundamental. Asimismo, la aplicación de validaciones de negocio y el uso de **borrado lógico** fortalecieron la integridad de los datos y la estabilidad del sistema.

- Desde una perspectiva teórica, el proyecto evidenció la importancia de conceptos como:

- La correcta definición de claves primarias y foráneas.
- El rol de la **unicidad** (ISBN).
- El uso de **constraints** de integridad.
- La separación entre lógica de negocio y acceso a datos (DAO + Service).
- La reutilización y mantenimiento favorecidos por OOP.

En términos prácticos, el sistema resultante es completamente funcional, robusto y con capacidad de crecimiento.
Como mejoras futuras, se identifican posibles extensiones como migrar a **Spring Boot**, integrar **logging profesional**, agregar pruebas automatizadas (**JUnit**), incorporar más entidades bibliotecarias y ofrecer una interfaz visual moderna (JavaFX o web).

En suma, el proyecto cumple ampliamente con los requisitos del TPI y sirve como un ejercicio sólido de integración entre teoría y práctica en el desarrollo de software.