Trabajo Final Integrador - Programación 2

Informe Técnico - Sistema de Gestión de Biblioteca

1. Integrantes del Equipo

Nombre	Rol en el Proyecto
Elvio Nahuel Romero Alani	Arquitectura y diseño de base de datos
Martín Ezequiel Toledo	Interface de usuario y testing
Gastón Zarate	Desarrollo de capa Service y validaciones
Abel Tomás Romero	Implementación de capa DAO y persistencia

2. Dominio Elegido y Justificación

Justificación de la elección del dominio: Libro → FichaBibliográfica
Elegimos el dominio de biblioteca por varias razones prácticas:

- **Relevancia real:** Los sistemas de gestión bibliotecaria son aplicaciones reales y tangibles que todos conocemos. Esto facilita entender los requisitos y validar que la solución tenga sentido.
- **Relación 1→1 natural:** La relación entre un libro y su ficha bibliográfica es genuinamente unidireccional y de uno a uno. Un libro tiene exactamente una ficha (o ninguna), y esa ficha existe específicamente para catalogar ese libro. No hay ambigüedad conceptual.
- **Complejidad adecuada:** El dominio de suficiente complejidad para demostrar el manejo de transacciones, validaciones y búsquedas, sin ser excesivamente complicado. Los campos opcionales y obligatorios muestran validaciones realistas.
- **Casos de uso claros:** Las operaciones CRUD tienen sentido intuitivo: crear libros con o sin ficha, actualizar información bibliográfica, buscar por diferentes criterios (autor, título, ISBN, idioma), y eliminar registros obsoletos.

3. Diseño y Decisiones Técnicas

3.1 Relación 1→1 Unidireccional

Implementación elegida: Foreign Key única en tabla “libro”

```
CREATE TABLE libro (
  ...
  ficha_bibliografica_id BIGINT UNIQUE,
  CONSTRAINT fk_libro_ficha FOREIGN KEY (ficha_bibliografica_id)
    REFERENCES ficha_bibliografica(id)
    ON DELETE CASCADE
);
```

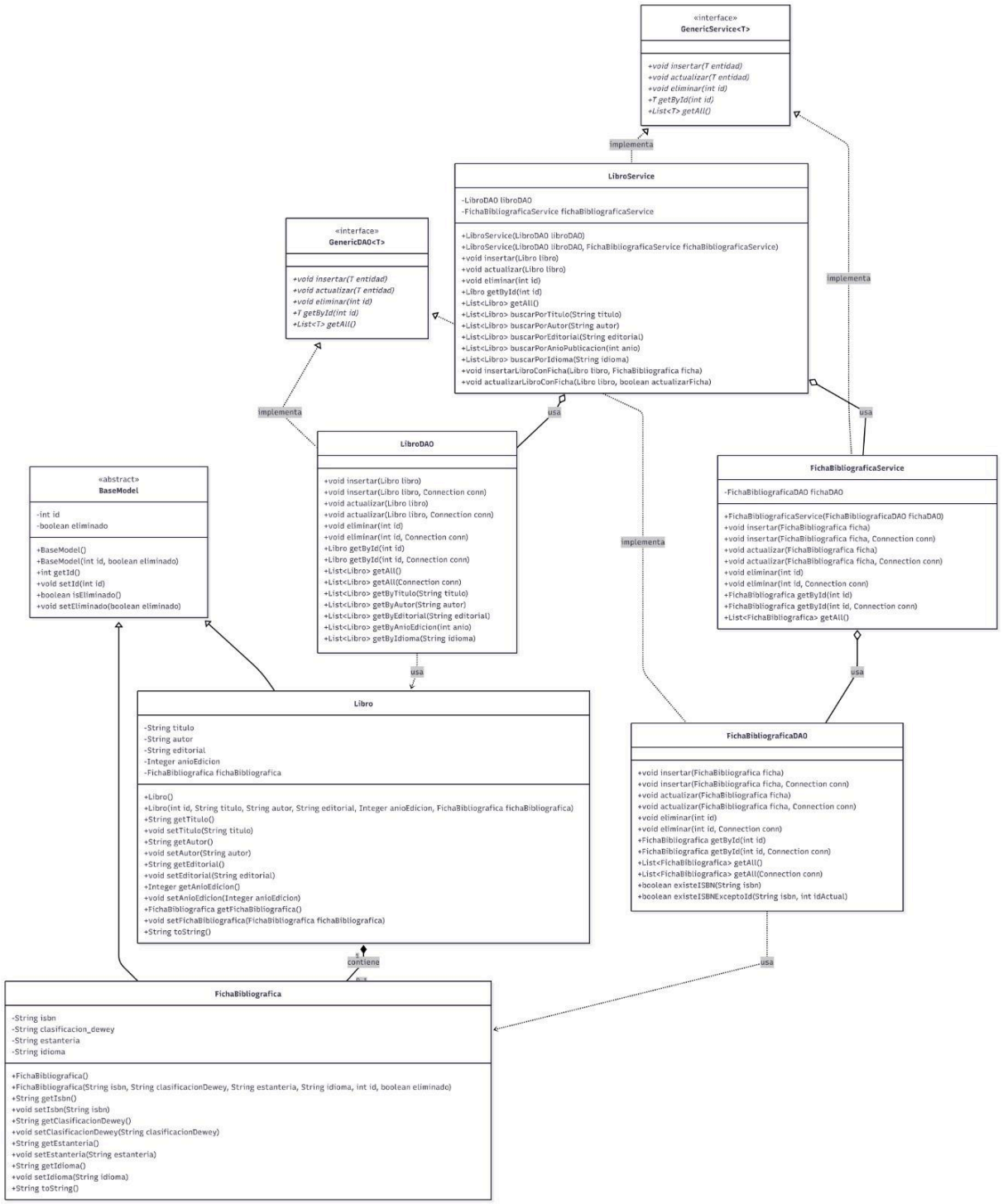
Razones de la decisión:

- **UNIQUE constraint:** Garantiza a nivel de base de datos que ninguna ficha pueda asociarse a más de un libro, cumpliendo estrictamente la relación 1→1.
- **ON DELETE CASCADE:** Si se elimina una ficha (físicamente), el libro asociado también se elimina. Esto mantiene la integridad referencial automáticamente.

- **Nullable FK:** Permite crear libros sin ficha inicialmente, dando flexibilidad operativa. Se puede añadir la ficha después mediante actualización.
- **Dirección unidireccional:** La clase `Libro` tiene una referencia a `FichaBibliografica`, pero no viceversa. La ficha no "conoce" a qué libro pertenece en el modelo de objetos.

Alternativa descartada: Clave primaria compartida (PK de ficha = FK a libro). Se descartó porque es menos flexible: obligaría a crear ambas entidades simultáneamente siempre, eliminando la posibilidad de crear un libro sin ficha.

3.2 Diagrama UML de Clases



4. Arquitectura en Capas

4.1 Paquete `config/`: Responsable de la gestión de conexiones y transacciones. Sus clases principales:

- **DatabaseConnection:** Singleton que proporciona conexiones JDBC. Lee configuración de propiedades del sistema (`Ddb.url`, `-Ddb.user`, `-Ddb.password`). Valida parámetros en bloque estático al cargar la clase.
- **TransactionManager:** Implementa `AutoCloseable` para uso con try-with-resources. Encapsula `setAutoCommit(false)`, `commit()` y `rollback()`. Garantiza que si algo falla, se revierte automáticamente en el `close()`.

4.2 Paquete `dao/`: Responsable de la persistencia de datos con JDBC puro (sin ORM). Sus características:

- **Doble set de métodos:** Cada DAO tiene métodos con conexión propia (para uso simple) y métodos que aceptan `Connection` externa (para transacciones).
- **PreparedStatement:** Todas las queries usan `PreparedStatement` para prevenir SQL injection.
- **Soft delete:** El método `eliminar()` hace `UPDATE SET eliminado = TRUE`, no `DELETE`. Las consultas se filtran con `WHERE eliminado = FALSE`.
- **Validaciones específicas:** `FichaBibliograficaDAO` incluye métodos como `existeISBN()` para validar unicidad antes de insertar.

4.3 Paquete `main/`: Responsable de la interfaz de usuario por consola. Sus clases son:

- **Main:** Punto de entrada, crea `AppMenu` y ejecuta.
- **AppMenu:** Controla el flujo de navegación entre menús.
- **MenuDisplay:** Renderiza los menús (separación de concerns).
- **MenuHandler:** Procesa las acciones del usuario, captura datos, invoca servicios y maneja excepciones. Incluye validaciones de entrada (con reintentos hasta obtener datos válidos).
- **TestConexion:** Utilidad para verificar la conectividad con la BD.

4.4 Paquete `models/`: Responsable de la representación de entidades del dominio. Sus clases principales:

- **BaseModel:** Atributos comunes (`id`, `eliminado`) para todas las entidades. Implementa soft delete.
- **Libro:** Campos obligatorios (título, autor) y opcionales (editorial, año). Contiene referencia a FichaBibliografica (puede ser null).
- **FichaBibliografica:** Todos los campos opcionales (ISBN, clasificación Dewey, estantería, idioma). Permite flexibilidad en el catalogado.

4.5 Paquete `Service/`: Responsable de la lógica de negocio, validaciones y orquestación de transacciones. Sus funciones principales:

- **Validaciones previas:** Antes de llamar al DAO, el Service valida formatos, longitudes máximas, campos obligatorios, y rangos válidos.
- **Normalización:** Convierte todos los textos a mayúsculas con `toUpperCase()` para uniformidad en búsquedas.
- **Transacciones complejas:** Métodos como `insertarLibroConFicha()` coordinan operaciones en múltiples tablas:
    1. Inicia transacción
    2. Inserta ficha (obtiene ID generado)
    3. Asocia ficha al libro
    4. Inserta libro
    5. Commit si todo OK, rollback automático si falla

5. Persistencia y Transacciones

5.1 Estructura de Base de Datos

- **Base de datos:** `dbtpi3` (MySQL con charset utf8mb4)
- **Tablas:**

```
ficha_bibliografica (
   id BIGINT AUTO_INCREMET PRIMARY KEY,
   eliminado BOOLEAN NOT NULL DEFAULT FALSE,
   isbn VARCHAR(17) UNIQUE,
   clasificacion_dewey VARCHAR(20),
   estanteria VARCHAR(20),
   idioma VARCHAR(30)
)

libro (
   id BIGINT AUTO_INCREMENT PRIMARY KEY,
   eliminado BOOLEAN NOT NULL DEFAULT FALSE,
   titulo VARCHAR(150) NOT NULL,
   autor VARCHAR(120) NOT NULL,
   editorial VARCHAR(100),
   anio_edicion INT,
   ficha_bibliografica_id BIGINT UNIQUE,
   CONSTRAINT fk_libro_ficha FOREIGN KEY (ficha_bibliografica_id)
      REFERENCES ficha_bibliografica(id)
      ON DELETE CASCADE
)
```

Índices implícitos: Las claves primarias y el constraint UNIQUE crean índices automáticos para optimizar búsquedas por ID e ISBN.

5.2 Orden de Operaciones en Transacciones

Caso: Crear libro con ficha (operación más compleja).

```
1. TransactionManager.startTransaction()
└─ setAutoCommit(false)

2. FichaBibliograficaService.insertar(ficha, conn)
├─ Validar ISBN único
├─ Normalizar campos
└─ FichaBibliograficaDAO.insertar(ficha, conn)
        └─ INSERT INTO ficha_bibliografica ...
        └─ Obtener ID generado → ficha.setId(generatedId)

3. libro.setFichaBibliografica(ficha)

4. LibroService (validaciones internas)
├─ Validar título/autor obligatorios
└─ Normalizar campos

5. LibroDAO.insertar(libro, conn)
└─ INSERT INTO libro (... ficha_bibliografica_id) VALUES (... ?, ...)    // El ? es ficha.getId()

6. TransactionManager.commit()
└─ conn.commit()
└─ setAutoCommit(true)

Si CUALQUIER paso falla: TransactionManager.close() → rollback automático
```

Ventajas del enfoque:

- **Atomicidad:** O se crea todo o no se crea nada. No quedan fichas huérfanas.
- **Manejo automático:** Try-with-resources garantiza rollback incluso si hay exception no capturada.
- **Reutilización de conexión:** La misma `Connection` se usa para todas las operaciones, manteniendo la transacción activa.

5.3 Puntos de Commit y Rollback

- **Commit explícito:** Después de completar todas las operaciones exitosamente en `LibroService.insertarLibroConFicha()` y `actualizarLibroConFicha()`.
- **Rollback automático:**
  - Si alguna validación falla (IllegalArgumentException).
  - Si hay error de SQL (SQLException).
  - Si hay cualquier Exception no capturada.
  - El `TransactionManager.close()` detecta transacción activa y hace rollback.

6. Validaciones y Reglas de Negocio

6.1 Validaciones en FichaBibliograficaService

- **ISBN:**
  - **Formato:** 10 o 13 dígitos (con o sin guiones).
  - **Regex:** `\d{10}|\d{13}` (sin contar guiones).
  - **Unicidad:** Verificado con query antes de insertar/actualizar.
  - **Ejemplo válido:** `978-3-16-148410-0`.
- **Clasificación Dewey:**
  - Opcional, con un máximo de 20 caracteres.
  - **Formato flexible:** `\d{1,3}(\.\d{1,2})?`.
  - **Ejemplos:** `005.1`, `863.64`, `100`
- **Estantería e Idioma:**
  - Opcionales.
  - **Validación simple:** longitud máxima.

6.2 Validaciones en LibroService

- **Título y Autor:**
  - Obligatorios (no pueden ser null ni vacíos).
  - **Longitudes máximas:** 150 y 120 caracteres respectivamente.
- **Año de Edición:**
  - Opcional
  - **Si se especifica:** debe estar entre 1000 y el año actual.
  - **Validación dinámica:** `java.time.Year.now().getValue()`.
- **Editorial:** Opcional, con un máximo de 100 caracteres.

6.3 Normalización de Datos

Todos los campos de texto se normalizan antes de persistir con: `texto.trim().toUpperCase()`. Las **ventajas:** Búsquedas case-insensitive consistentes; elimina espacios en blanco accidentales; uniformidad en la visualización.

7. Pruebas Realizadas

7.1 Escenarios de Prueba Funcional

Caso 1: Crear libro con ficha (transacción exitosa)

- **Input:**
  - **Libro:** "CIEN AÑOS DE SOLEDAD" / "GABRIEL GARCÍA MÁRQUEZ".
  - **Ficha:** ISBN "978-3-16-148410-0", Dewey "863.64"
- **Resultado esperado:** Ambos creados, IDs asignados
- **Resultado obtenido:** Libro ID: 1, Ficha ID: 1
- **Verificación SQL:** `SELECT * FROM libro JOIN ficha_bibliografica`

Caso 2: Crear libro sin ficha

- **Input:** Libro sin asociar ficha
- **Resultado esperado:** Sólo se crea libro, ficha_bibliografica_id = NULL
- **Resultado obtenido:** Libro creado correctamente

Caso 3: Transacción con rollback (ISBN duplicado)

- **Input:** Intentar crear libro con ISBN ya existente
- **Resultado esperado:** Rollback, ningún registro insertado
- **Resultado obtenido:** Exception capturada, rollback automático
- **Verificación SQL:** `COUNT(*) sin cambios`

Caso 4: Búsqueda por criterios

- Búsqueda por autor "GARCÍA": 1 resultado
- Búsqueda por idioma "ESPAÑOL": 2 resultados
- Búsqueda por año 1967: 1 resultado

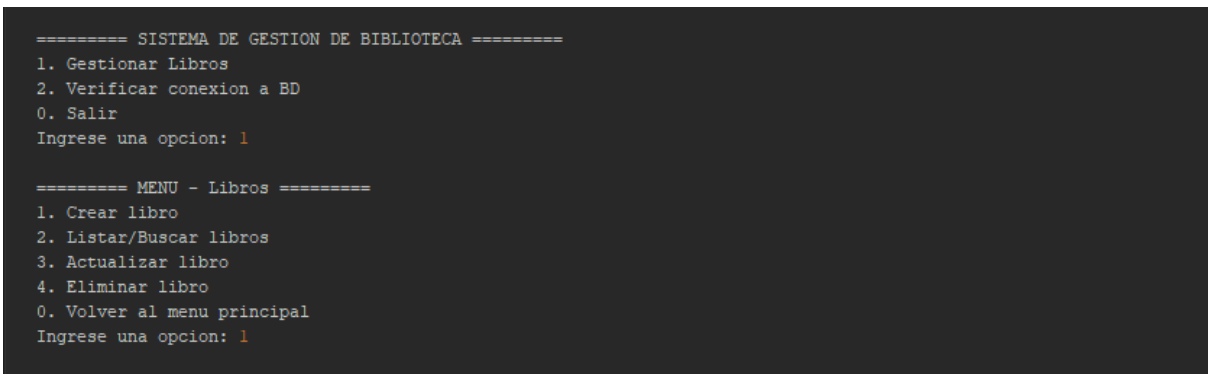
Caso 5: Actualización con transacción

- **Input:** Modificar libro y su ficha en una operación
- **Resultado esperado:** Ambos actualizados atómicamente
- **Resultado obtenido:** Transacción completada

Caso 6: Eliminación lógica

- **Input:** Eliminar libro ID 2
- **Verificación SQL:**
  - `SELECT * WHERE eliminado = FALSE`: no aparece
  - `SELECT * WHERE eliminado = TRUE`: aparece con flag
- **Resultado:** Soft delete funcional

7.2 Capturas de Menú



```
========= CREAR LIBRO =========
Titulo: Clean Code
Autor: Robert C. Martin
Editorial (Enter para omitir): Pearson Education
Anio de edicion (Enter para omitir): 2008
Desea crear una ficha bibliografica para el libro? (S/N): n
Libro Creado exitosamente con ID: 8
```

```
========= LISTAR/BUSCAR LIBROS =========
1. Listar todos
2. Buscar por autor
3. Buscar por titulo
4. Buscar por anio de publicacion
5. Buscar por idioma
0. Volver
Opcion: 1

========= RESULTADOS (1) =========

ID: 8
  Titulo: CLEAN CODE
  Autor: ROBERT C. MARTIN
  Editorial: PEARSON EDUCATION
  Anio: 2008
```

7.3 Consultas SQL Útiles

```
-- Ver todos los libros con sus fichas
SELECT l.id, l.titulo, l.autor, f.isbn, f.idioma
FROM libro l
LEFT JOIN ficha_bibliografica f ON l.ficha_bibliografica_id = f.id
WHERE l.eliminado = FALSE;

-- Verificar constraint de unicidad FK
SELECT ficha_bibliografica_id, COUNT(*)
FROM libro
WHERE eliminado = FALSE
GROUP BY ficha_bibliografica_id
HAVING COUNT(*) > 1;
-- Debe retornar 0 filas

-- Ver registros eliminados (auditoría)
SELECT * FROM libro WHERE eliminado = TRUE;
```

8. Conclusiones y Mejoras Futuras

8.1 Objetivos Cumplidos

- **Relación 1→1 unidireccional:** Implementada correctamente con FK única y validaciones en código.
- **Patrón DAO:** Separación clara entre persistencia y lógica de negocio.
- **Transacciones:** Manejo con commit/rollback mediante `TransactionManager`.
- **CRUD completo:** Todas las operaciones funcionando con soft delete.
- **Validaciones:** Reglas de negocio aplicadas consistentemente en la capa Service.
- **Búsquedas avanzadas:** Múltiples criterios implementados (título, autor, año, idioma).

8.2 Lecciones Aprendidas

- **Diseño de transacciones:** Usar `TransactionManager` con try-with-resources simplificó enormemente el manejo de errores. Evitamos fugas de recursos y rollbacks olvidados.
- **Doble set de métodos en DAOs:** Mantener métodos con conexión propia y métodos con conexión externa dio flexibilidad total: operaciones simples son directas, y transacciones complejas reutilizan la misma conexión.
- **Normalización temprana:** Aplicar `toUpperCase()` en el Service antes de persistir eliminó inconsistencias en búsquedas.
- **Validación en capas:** Validar en Service (lógica de negocio) y en BD (constraints) da doble protección contra datos inválidos.

8.3 Mejoras Futuras

- **Implementar logging:** Usar SLF4J + Logback para registrar operaciones y errores en archivos. Facilitaría el debugging en producción.
- **Pool de conexiones:** Integrar HikariCP para gestionar conexiones de forma eficiente en lugar de crear una nueva por operación.
- **Paginación:** Las búsquedas actuales retornan todos los resultados. Con muchos libros, implementar `LIMIT/OFFSET` mejoraría el rendimiento.
- **Caché:** Usar Caffeine o similar para cachear libros consultados frecuentemente, reduciendo hits a la BD.
- **Testing automatizado:** Agregar JUnit 5 con tests unitarios para Services y tests de integración con base de datos embebida (H2).
- **Interfaz gráfica:** Migrar de consola a JavaFX o web (Spring Boot + Thymeleaf) para mejorar usabilidad.
- **Auditoría completa:** Guardar quién y cuándo modificó cada registro (campos `created_by`, `modified_by`, `modified_at`).
- **Búsqueda fulltext:** Implementar índices FULLTEXT en MySQL para búsquedas más potentes y rápidas en campos de texto largo.

9. Herramientas y Referencias Utilizadas

- **Lenguajes y Frameworks:**
  - Java 17+ y MySQL 8.0
  - JDBC Driver (mysql-connector-j 8.0+)
- **IDEs y Herramientas:**
  - NetBeans para Java y Dbeaver para diseño de base de datos
  - Git/GitHub para control de versiones
  - Mermaid para el diagrama UML
- **Referencias consultadas:**
  - Documentación oficial de Oracle JDBC: https://docs.oracle.com/en/java/javase/21/docs/api/java.sql/
  - MySQL Reference Manual: https://dev.mysql.com/doc/
  - Patron DAO: https://www.oracle.com/java/technologies/dataaccessobject.html
- **Uso de IA:**
  - Claude (Anthropic) para consultas sobre buenas prácticas de transacciones JDBC y optimización de queries SQL.
  - ChatGPT: Sugerencias de mejoras de código y documentaciones.
  - GitHub Copilot para autocompletado de código repetitivo (getters/setters, validaciones similares, documentación).