Programación II

Trabajo Practico Integrador

Alumnos:

- Salinas, Santiago Raúl;
- Sanabria, Ezequiel Franco;
- Tozzi, Gonzalo;
- Vallejos, Emiliano Víctor.

Comisión: 7 - 9

Profesor: Cinthia Rigoni.

Fecha: 17/11/2025

Índice

Elección del dominio y justificación

Para este trabajo integrador se eligió el dominio ***Gestión de Pedidos y Envíos***, un escenario muy habitual en sistemas reales de e-commerce, logística y compras online.

La elección se justifica por los siguientes motivos:

1. Es un dominio sencillo, pero con la complejidad suficiente para aplicar conceptos de integridad, relaciones, validaciones y transacciones.
2. El flujo Pedido → Envío refleja un caso de negocio real.
3. Permite implementar naturalmente una relación **1→1** entre entidades.
4. Resulta adecuado para demostrar manejo de errores, reglas de negocio y consistencia transaccional.

Diseño del modelo y decisiones clave

Relación 1→1

El trabajo exige una relación 1→1 unidireccional explícita entre dos entidades. En este caso:

1. Pedido contiene una referencia a Envio (private Envio envio;)
2. Envio referencia a Pedido únicamente desde la base de datos (campo id_pedido).

Justificación del 1→1:

1. Un Pedido solo puede tener un Envío asociado.
2. Un Envío siempre pertenece a un único Pedido.
3. Se evita duplicidad y se mantiene cohesión en el dominio.

Implementación de la relación: FK única vs PK compartida

Existen dos formas de implementar un 1→1 en bases relacionales:

1. PK compartida (la PK de Envío es también FK de Pedido).

2. FK única (la tabla Envío tiene una FK id_pedido marcada como UNIQUE).

Decisión adoptada:

Se eligió FK única, porque:

1. Es más simple de leer, mantener y debuggear.
2. Mantiene independencia de identidad entre Pedido.id y Envio.id.
3. Permite inserciones y consultas más claras.
4. Favorece el modelo DAO/Service implementado.

Fragmento relevante de SQL:

CONSTRAINT uk_envio_id_pedido UNIQUE (id_pedido)
CONSTRAINT fk_envio_pedido FOREIGN KEY (id_pedido)
REFERENCES Pedido(id) ON DELETE CASCADE

Este constraint garantiza de forma nativa que un Pedido no puede tener más de un Envío, cumpliendo estrictamente el 1→1.

Arquitectura por capas

El proyecto se estructuró utilizando una arquitectura clara en capas, separando responsabilidades:

- config/ → conexión a la base (DatabaseConnection)
- model/ → entidades Pedido y Envio
- dao/ → acceso a datos (PreparedStatement, CRUD)
- service/ → reglas de negocio + transacciones + validaciones
- app/ → AppMenu, interacción con usuario

Roles de cada capa

Model (Entidad)

- Representan objetos del dominio.
- Contienen atributos, constructores, getters/setters y toString().
- Incluyen campo eliminado para bajas lógicas.

DAO

- Implementan CRUD usando JDBC puro.
- Encapsulan SQL y devuelven Optional o listas.
- Aceptan una Connection externa para participar de transacciones.

Service

- Contienen validaciones y reglas de negocio.
- Manejan transacciones (setAutoCommit(false), commit(), rollback()).
- Impiden la violación del modelo 1→1 (un Envío por Pedido).
- Manejan errores de negocio y de integridad.

AppMenu

- Presenta el menú de consola.
- Maneja entradas del usuario.
- Invoca los métodos apropiados de los services.
- Implementa CRUD completo de ambas entidades.
- Incluye una opción especial de operación transaccional.

Persistencia y estructura de la base de datos

La base de datos se diseñó en MySQL.

Tablas principales:

- Pedido

- Envio

Ambas con:

- id BIGINT AUTO_INCREMENT
- eliminado BOOLEAN
- constraints de unicidad y check
- relación 1→1 vía FK unique

Orden de operaciones

- Siempre se crean primero los Pedidos.
- Envíos dependen del Pedido y se eliminan en cascada (ON DELETE CASCADE).
- Para operaciones compuestas (Pedido + Envío) se utiliza transacción.

Transacciones

En *PedidoService.insertar(Pedido p):*

```
conn.setAutoCommit(false);

long idPedido = pedidoDao.crear(pedido, conn);

if (pedido.getEnvio() != null) {
   envioDao.crear(pedido.getEnvio(), conn);
}

conn.commit();
```

En caso de error:

```
conn.rollback();
```

Esto asegura atomicidad

Validaciones y reglas de negocio

En la capa Service se aplican todas las validaciones obligatorias:

Validaciones de Pedido:

- Número obligatorio
- Nombre del cliente obligatorio
- Total ≥ 0
- Estado válido (NUEVO / FACTURADO / ENVIADO)

Validaciones de Envío:

- Tracking obligatorio
- Empresa válida
- Tipo válido
- Estado válido

Reglas de negocio:

- Un Pedido no puede tener más de un Envío.
  Se controla en BD (unique) y en Service.
- Las operaciones mixtas (Pedido + Envío) son transaccionales.
- Eliminación lógica: eliminado = true.

Pruebas realizadas

Se realizaron pruebas manuales desde el menú principal.

CRUD Pedido

- Crear Pedido
- Listar
- Buscar por ID
- Actualizar
- Eliminar (lógico)

- Verificar que eliminados no aparecen en listados.

CRUD Envío

- Crear Envío asociado a Pedido
- Listar todos
- Buscar por ID
- Actualizar
- Eliminar lógico

Prueba del 1→1

Intento de crear un segundo Envío para el mismo Pedido → el sistema devuelve error:

Este pedido ya tiene un envío asociado.

Prueba de transacción (Pedido + Envío)

Opción del menú:

"Crear Pedido + Envío (Transacción)"

Se probó:

- Completar datos
- Ingresar un tracking duplicado para forzar el error

Resultado correcto:

ERROR — Se realizó rollback, nada fue guardado

Conclusiones y mejoras futuras

Conclusiones

- Se implementó un sistema completo de gestión de Pedidos y Envíos.
- La relación 1→1 se garantizó tanto a nivel lógico como de base de datos.

- Se aplicó arquitectura por capas, validaciones, restricciones y transacciones.
- Se demostró atomicidad mediante implementación real de rollback.
- El menú permite operar la totalidad del modelo de forma sencilla.

Mejoras futuras

- Implementar campos adicionales (dirección de envío, teléfono, métodos de pago).
- Agregar paginación en listados.
- Implementar autenticación de usuarios.
- Agregar logs y manejo de excepciones más detallado.

Fuentes y herramientas utilizadas

Herramientas

- Java 17
- NetBeans
- MySQL y MySQL Workbench
- JDBC (PreparedStatement)
- Git y GitHub
- IA (ChatGPT) utilizada como *asistente de organización, explicación y redacción*, no como generador directo del código final.

Documentación consultada

- Documentación oficial de Java JDBC
- Documentación de MySQL
- Material de cátedra de Programación II

Link Video Youtube

https://www.youtube.com/watch?v=NGb2ze71KVw


UTN
TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA