TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

TRABAJO FINAL INTEGRADOR

Programación II

Alumnos:

Facundo Auciello (Comisión Ag25-2C 07)

Ayelen Etchegoyen (Comisión Ag25-2C 07)

Alexia Rubin (Comisión Ag25-2C 05)

María Victoria Volpe (Comisión Ag25-2C 09)

Fecha de Entrega: 17 de noviembre de 2025

Enlace al video:

https://www.youtube.com/watch?v=6gJJFeiNgSE

Enlace al repositorio de github:

https://github.com/alx1as/TFI_programacion2

1. Introducción

El presente trabajo integrador desarrolla un sistema sencillo para gestionar usuarios y sus credenciales de acceso. La aplicación permite crear, buscar, actualizar, listar y eliminar tanto usuarios como credenciales, además de realizar búsquedas específicas por email o por nombre de usuario.

El proyecto se desarrolló en Java utilizando JDBC y MySQL, y organizando el código en capas. Se trabajó con operaciones CRUD, uso de menús por consola y un modelo donde cada usuario tiene asociada una única credencial. También se manejaron transacciones para asegurar que ciertas operaciones se realicen de manera consistente.

1.b Integrantes y Roles

El trabajo fue realizado por un equipo de cuatro integrantes de la Tecnicatura Universitaria en Programación a distancia. La organización se realizó por carpetas siguiendo la arquitectura por capas solicitada. Cada integrante asumió un rol principal dentro de esa estructura:

Alexia Rubin: responsable del diseño del dominio, elaboración del diagrama UML de clases y definición de la relación 1→1 entre Usuario y CredencialAcceso. Implementó el paquete config (DatabaseConnection) y desarrolló el paquete models (Usuario, CredencialAcceso).

María Victoria Volpe: responsable de la capa de persistencia (dao). Implementó UsuarioDao, CredencialAccesoDao y la interfaz genérica GenericDao, definiendo todas las operaciones CRUD con JDBC y PreparedStatement. También realizó pruebas de acceso a datos para validar consultas y comportamiento de la base.

Ayelén Etchegoyen: responsable de la capa de servicios (services). Implementó UsuarioService, CredencialService y GenericService, encargándose de la lógica de

negocio, validaciones, coordinación de flujos complejos y manejo de transacciones. Utilizó TransactionManager para gestionar correctamente commit y rollback en operaciones compuestas, asegurando la consistencia entre Usuario y CredencialAcceso.

Facundo Auciello: responsable de la capa de presentación (main). Implementó Main, AppMenu, MenuHandler y MenuDisplay, desarrollando todo el sistema de interacción por consola. Diseñó el flujo de opciones, el ingreso de datos por parte del usuario y la comunicación con los servicios. Su trabajo permitió ejecutar completamente las funcionalidades del sistema desde el menú.

Todos los integrantes contribuyeron a la creación, revisión y pruebas de los scripts SQL de la base de datos.

---

2. Elección del dominio y justificación

El dominio elegido para este trabajo fue **Usuario → CredencialAcceso**. Nos pareció adecuado porque es un caso simple y muy común en sistemas reales: casi cualquier aplicación necesita registrar usuarios y manejar sus datos de acceso. En este caso, el dominio se adapta naturalmente a una relación 1→1, ya que cada usuario tiene exactamente una credencial asociada.

También resultó un buen ejemplo para aplicar las técnicas vistas en clase: separación por capas, uso de JDBC, validaciones básicas, baja lógica, manejo de transacciones y operaciones CRUD completas.

3. Modelo de Base de Datos y Decisiones de Diseño

Para el diseño se elaboró un diagrama UML que representa las entidades principales del proyecto y la relación 1→1 entre ellas. El modelo está formado por tres clases:

Base (superclase)
Contiene los atributos comunes a todas las entidades: **id** y **eliminado.** Ambas clases del dominio heredan de esta estructura para aplicar la baja lógica solicitada en el trabajo.

Usuario
Define la información del usuario (Nombre, Apellido, Edad, Email, Usuario). Incluye un atributo de tipo **CredencialAcceso,** lo que establece la relación unidireccional 1→1. Esto significa que desde un usuario se puede acceder a su credencial pero no al revés.

CredencialAcceso
Representa la credencial asociada al usuario. Contiene la fecha de creación, la contraseña y el idUsuario que asegura la relación con la clase Usuario.

Sobre esta base conceptual se estructuró la **base de datos usuariocredencial**, que respeta la misma relación y aplica las restricciones necesarias para mantener la integridad de los datos:

2.1. Tabla usuarios

- eliminado BOOLEAN para baja lógica.
- Restricción edad >= 18.
- Campos email y usuario con UNIQUE.

2.2. Tabla credencialAcceso

- Relación **1:1** con usuarios mediante id_usuario UNIQUE.
- contrasenia con restricción mínima de longitud.

- fecha_creacion almacenada como DATE.

Estas decisiones garantizan:

✔ integridad referencial
✔ control de datos inválidos
✔ estructura clara para CRUD + auditoría de accesos

LINK A UML:

https://github.com/alx1as/TFI_programacion2/blob/main/UML.pdf

---

3. Arquitectura por capas

Para organizar mejor el proyecto usamos una estructura por capas, dividiendo el código según la función de cada parte:

config
En esta carpeta dejamos todo lo relacionado con la conexión a la base de datos. DatabaseConnection abre la conexión a MySQL y TransactionManager nos permite manejar commit y rollback cuando una operación necesita hacerse de forma más segura.

models
Aquí están las clases que representan nuestras entidades: Usuario, CredencialAcceso y Base. Las dos primeras heredan de Base para usar el campo eliminado. Además, Usuario tiene la referencia a su credencial para mantener la relación 1→1.

dao
Esta carpeta contiene las clases que se encargan de hablar directamente con la base de datos. Incluye GenericDao y los DAO concretos de usuario y credencial, donde hicimos los métodos para crear, listar, buscar, actualizar y eliminar usando JDBC.

services

En esta parte va la lógica “intermedia” del sistema. Los servicios reciben los datos del menú, hacen las validaciones necesarias y llaman a los DAO.
También usan el TransactionManager cuando hace falta trabajar varias cosas juntas (por ejemplo, crear un usuario y su credencial).

main

Es la capa con la que interactúa el usuario. Acá está el menú por consola: Main, AppMenu, MenuDisplay y MenuHandler.Desde acá se eligen las opciones y se ejecutan todas las funciones del sistema.

4. Persistencia y Transacciones

La persistencia del sistema se manejó con MySQL y JDBC. Para esto creamos la base de datos usuariocredencial, donde están las tablas usuarios y credencialAcceso. Cada tabla refleja el modelo del UML y las reglas que definimos en el diseño.

En los DAO se hicieron todos los métodos CRUD usando PreparedStatement, para evitar problemas de seguridad y trabajar con consultas parametrizadas. Desde allí se insertan, actualizan, buscan y eliminan (de forma lógica) los datos.

Para las operaciones más simples, cada acción se ejecuta directamente. Pero cuando necesitamos que dos cosas se hagan juntas de forma obligatoria (por ejemplo, crear un usuario y su credencial) se usa el TransactionManager. Esto nos permite desactivar el autocommit, ejecutar varios pasos y finalmente decidir si todo se confirma (commit) o se deshace (rollback) si algo falló.

Esta forma de trabajar garantiza que la relación 1→1 se mantenga consistente y que no queden datos incompletos en la base en caso de error. También facilita el registro de accesos, ya que cada ingreso queda guardado con su fecha y hora correspondiente.

5. Validaciones y reglas de negocio

El sistema incluye validaciones básicas en la capa de servicios para evitar datos incorrectos. Se controla que los campos obligatorios no estén vacíos, que la edad sea válida y que el ID ingresado exista antes de realizar cualquier operación.

La base de datos agrega sus propias reglas: email y usuario son únicos, la edad mínima debe cumplirse y la relación 1→1 se asegura con id_usuario como campo único en credencialAcceso. También se valida que la contraseña tenga una longitud mínima y que no se cree una credencial para un usuario inexistente.

La baja lógica se maneja con el campo eliminado, lo que permite mantener registros sin borrarlos definitivamente.

6. Pruebas realizadas

Para comprobar el funcionamiento del sistema se realizaron distintas pruebas desde el menú por consola y también directamente sobre la base de datos. Se probaron todas las operaciones CRUD tanto de usuarios como de credenciales: creación, búsqueda por ID, actualización, listado y eliminación lógica.

También se verificaron las búsquedas por email y por username, comprobando que devolvieran el usuario correcto. Durante las pruebas forzamos algunos errores comunes (IDs inexistentes, emails duplicados, campos vacíos, credenciales sin usuario válido) para confirmar que las validaciones y las reglas de la base de datos funcionaran como se espera.

Además, se utilizaron consultas SQL para revisar el contenido de las tablas después de cada operación, especialmente para comprobar la relación 1→1, el campo eliminado y el registro de accesos con fecha y hora.

7. Conclusiones y mejoras futuras

El desarrollo de este trabajo nos permitió aplicar en conjunto los contenidos de Programación 2 y de Bases de Datos, combinando el diseño SQL con la implementación en Java. Esto nos ayudó a entender mejor cómo se conectan ambos mundos: cómo un modelo pensado en SQL se transforma luego en clases, servicios y operaciones concretas dentro del programa.

En cuanto al proyecto logramos implementar un sistema completo y funcional, con CRUD, validaciones, transacciones y una relación 1→1 bien manejada. El trabajo en capas también nos ayudó a ordenar el código y a entender mejor el rol de cada parte del sistema.

Como mejoras futuras, sería posible agregar un manejo más seguro de contraseñas, ampliar el registro de accesos, incorporar más entidades o incluso crear una interfaz gráfica para reemplazar el menú por consola.

8. Fuentes y herramientas utilizadas

Para realizar este trabajo utilizamos Java como lenguaje principal y MySQL como gestor de base de datos. Trabajamos con JDBC para la conexión y ejecución de consultas, y organizamos el proyecto en NetBeans siguiendo la estructura por capas solicitada. También usamos GitHub para llevar el control de versiones y compartir el proyecto dentro del grupo.

Como apoyo consultamos la documentación oficial de Java, MySQL y JDBC, además del material teórico y práctico brindado en la cursada de Programación II y del Trabajo Integrador de Bases de Datos, lo que nos permitió relacionar ambos contenidos y entender mejor cómo se integran SQL y Java en un sistema real.

Finalmente, se emplearon recursos en línea y herramientas de inteligencia artificial únicamente como ayuda puntual para aclarar dudas y mejorar la redacción de la documentación, sin reemplazar el desarrollo propio del código.