


# Trabajo Final Integrador
# Aplicación Java con relación 1→1 unidireccional + DAO + MySQL

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

## Programación 2

## Alumnos - Comision 3

Tahiel Noé Heinze – tahielnoeheinze@gmail.com - 45237386

Tobias Leiva tobiasleivaa@gmail.com - 47622125

Genaro Luna - genaroluna2808@gmail.com - 44369231

## Profesor

Ariel Enferrel

## Tutor

Tomás Ferro

## Fecha de Entrega

17 de Noviembre de 2025

## Índice



---

## 1. Introducción:

***Desarrollar una aplicación en Java que modele dos clases relacionadas mediante una asociación unidireccional 1 a 1 (la clase "A" referencia a la clase "B"), persistiendo datos en una base relacional mediante JDBC y el patrón DAO, con operaciones transaccionales (commit/rollback) y menú de consola para CRUD.***

***Consignas generales***

***• Lenguaje: Java (recomendado 21).***
***• Persistencia: JDBC (sin ORM) con MySQL.***
***• Patrón de diseño: DAO y capa Service.***
***• Código limpio, legible, con manejo de excepciones en todas las capas.***
***• Relación 1→1 unidireccional: sólo la clase "A" contiene el atributo que referencia a "B".***
***• Equipos de 4 personas, sin excepción***

## 2. Integrantes y sus roles:

**TAHIEL NOÉ HEINZE**

***Rol principal:*** Diseño inicial y base de datos (1 y 3).

***Aportes:***

- Elaboró el Diagrama UML de clases, que sirvió como guía para toda la implementación.
- Diseñó y creó la base de datos en MySQL, definiendo tablas y relaciones.
- Redactó los scripts SQL (crear_base.sql y datos_prueba.sql) para la creación de tablas y carga de datos de prueba.
- Desarrolló las clases de entidades base: Usuario y CredencialAcceso, con sus atributos y métodos.
- Coordinó la organización y redacción del informe final en PDF.

**TOBIAS LEIVA**

***Rol principal:*** Entidades y capa DAO (2 y 4).

***Aportes:***

- Implementó la interfaz genérica GenericDao para operaciones CRUD.
- Desarrolló las clases concretas UsuarioDao y CredencialAccesoDao.
- Programó las consultas SQL utilizando PreparedStatement para mayor seguridad.
- Realizó pruebas de integridad referencial, verificando la relación 1 a 1 entre Usuario y CredencialAcceso.
- Colaboró en la redacción del informe final en PDF.

**GENARO LUNA**

***Rol principal:*** Lógica de negocio y menú de consola (5 y 6).

***Aportes:***

- Implementó la capa Service (GenericService, UsuarioService, CredencialAccesoService).
- Desarrolló el manejo de transacciones con commit y rollback.
- Incorporó validaciones de reglas de negocio, como formato de email y unicidad de usuarios.
- Programó el menú de consola (AppMenu) para la interacción con el sistema.
- Configuró la conexión a la base de datos y colaboró en el README y documentación general.
- Participó en la redacción del informe final en PDF.

## 3. Elección del dominio y justificación:

***Dominio elegido:*** "*Usuario → CredencialAcceso*".

Como equipo seleccionamos el dominio "*Usuario → CredencialAcceso*" por su aplicabilidad práctica en sistemas reales. Este dominio está presente en prácticamente cualquier aplicación que requiera autenticación, desde páginas web hasta aplicaciones móviles, lo que nos permitió trabajar con un caso de uso concreto y relevante.

Esta elección de dominio nos dio la oportunidad de entender cómo separar la información básica del usuario de sus datos sensibles de autenticación, a su vez la relación 1 a 1 unidireccional resultó natural e intuitiva: cada usuario puede tener una credencial asociada, pero la credencial no necesita conocer al usuario, simplificando el diseño y reflejando cómo funcionan realmente los sistemas de login.

## 4. Decisiones claves + UML:

***Relación 1→1 Unidireccional:***

Decidimos que la relación sea unidireccional desde "*Usuario"* hacia *"CredencialAcceso"* porque cuando trabajamos con un usuario necesitamos acceder rápidamente a sus credenciales para validar su autenticación. A su vez las credenciales por sí solas no es necesario que conozcan todos los datos del usuario, solamente deben de almacenar los datos de autentificación del usuario.

***FK única vs PK compartida:***

Optamos por utilizar una clave foránea única en la tabla *"credencial_acceso"* que referencia a *"usuario(id)"*. Tomamos esta decisión por la flexibilidad que nos otorga a la hora del registro de diferentes usuarios, como por ejemplo, la creación de una cuenta institucional donde un administrador puede crearle una una cuenta al usuario con sus datos básicos (nombre, username, email) y asignándole una contraseña/credencial temporal basándose en el DNI o una clave genérica al azar, la cual el usuario/estudiante luego de ingresar al sistema con esa clave pueda modificarla por una personalizada.
El campo "*usuario_id"* con restricción *"UNIQUE"* garantiza que cada credencial pertenezca a un único usuario, mientras que al permitir valores *"NULL"* posibilitamos la existencia temporal de usuarios sin credenciales durante procesos de registro por etapas.

***Baja lógica "eliminado":***

Implementamos baja lógica mediante el campo booleano *"eliminado"* en ambas tablas en lugar de realizar eliminaciones físicas *(DELETE)*.
Esta implementación nos permite recuperar cuentas eliminadas, y mantener los registros históricos.
Cuando una cuenta es eliminada/deshabilitada simplemente colocamos *"eliminado = TRUE"* lo cual permite excluirlo a la hora de realizar consultas colocar *"WHERE eliminado = FALSE"* conservando sus datos por si el usuario desea reactivar su cuenta o para la empresa que pueda realizar una auditoría.

***Separación de Usuario y CredencialAcceso:***

Decidimos separar la información de "*Usuario"* y *"CredencialAcceso"* en dos tablas distintas en lugar de tener una única tabla con todos los campos, ya que esta

separación responde al principio de responsabilidad única y a consideraciones de seguridad. Donde La tabla "*Usuario"* contiene datos de perfil (username, email, estado), mientras que *"CredencialAcceso"* almacena exclusivamente información sensible de autenticación (hash de contraseña, sal criptográfica).
Esta arquitectura nos facilita en un futuro el poder aplicar diferentes niveles de seguridad, como la encriptación de datos en la tabla de *"CredencialAcceso"* o el poder almacenar esta misma en otro servidor sin afectar el acceso a los datos generales del usuario.

***Campos con la restricción UNIQUE:***

Establecimos restricciones *"UNIQUE"* en tres campos estratégicos: *"usuario.username", "usuario.email"* y *"credencial_acceso.usuario_id"*. Donde los primeros dos garantizan que no puedan existir usuarios duplicados en el sistema, algo esencial para la identificación única de cada usuario. Y el tercero es fundamental para asegurar la relación 1 a 1, el cual impide que la misma credencial sea compartida por múltiples usuarios.

## 5. Arquitectura por capas:

**Paquete config/**
**DatabaseConnection.java**
**Paquete entities/**
**CredencialAcceso.java**
**Usuario.Java**
**Base.java**
**Paquete dao/**
**GenericDao.java**
**CredencialAccesoDao.java**
**UsuarioDao.java**
**Paquete service/**
**CredencialAccesoService.java**
**GenericService.java**
**UsuariService.java**
**Paquete main/**
**AppMenu.java**
**Main.java**

## 6. Persistencia: estructura de la base, orden de operaciones y transacciones:

**Estructura de la base de datos:**

- Tabla *"usuario"*.

- Tabla *"CredencialAcceso"*.
- Relación 1→1 mediante:
    - *credencial_acceso.usuario_id (UNIQUE) → usuario.id*
- *ON DELETE CASCADE* el cual garantiza la consistencia.
- Uso de *"eliminado"* para baja lógica.

**Orden de operaciones:**

- Al crear el usuario completo con su credencial, primero se ejecuta la creación del usuario y luego la credencial, seguido actualizamos y hacemos un commit para guardar los cambios en la conexión y del lado del DAO.

```
public void crearUsuarioCompleto(Usuario user, CredencialAcceso credencial) throws Exception {
    // 1. Validaciones primero
    validarUsuario(user);
    if (credencial == null) {
        throw new IllegalArgumentException("La credencial no puede ser nula");
    }

    Connection conn = null;
    try {
        // 2. Iniciar la transacción manualmente
        conn = DatabaseConnection.getConnection();
        conn.setAutoCommit(false);
        // 3. Ejecutar las operaciones (pasando la MISMA conexión)
        // Paso A: Crear el usuario (esto genera el ID)
        usuarioDao.crear(user, conn);
        // Paso B: Asignar el ID generado a la credencial
        credencial.setId(user.getId());
        // Paso C: Crear la credencial
        credencialDao.crear(credencial, conn);
        // Paso D: Actualizar el usuario para vincular credencial_id
        user.setCredencial(credencial);
        usuarioDao.actualizar(user, conn);
        // 4. Si todo salió bien, confirmar
        conn.commit();
```

**Manejo de commit y rollback:**

- Si se comete un error al momento de crear las crear el usuario y en medio de las 2 transacciones, se hace un chat y se ejecuta el rollback para no guardar los cambios:

```
    } catch (Exception e) {
        // 5. Si algo falló, revertir
        if (conn != null) {
            conn.rollback();
        }
        // Propagar el error
        throw new Exception("Error en la transacción 'crearUsuarioCompleto': " + e.getMessage(), e);
    } finally {
        // 6. Cerrar y restaurar
        if (conn != null) {
            conn.setAutoCommit(true);
            conn.close();
        }
    }
}
```

El rollback se encuentra tanto del lado de la creación de usuario y credenciales, como del lado del eliminado lógico del mismo ya que se elimina la credenciales asignada a ese usuario.

## 7. Validaciones y reglas de negocio:

### Validaciones implementadas:

- *"username"* no puede estar vacío.
- *"email"* debe respetar un formato válido.
- Debe existir unicidad de *"email"* y *"username"*.
- No se permite asignar más de una credencial al mismo usuario.
- No se pueden crear credenciales sin "*hash_password*".
- No se puede eliminar físicamente un registro → sólo baja lógica.

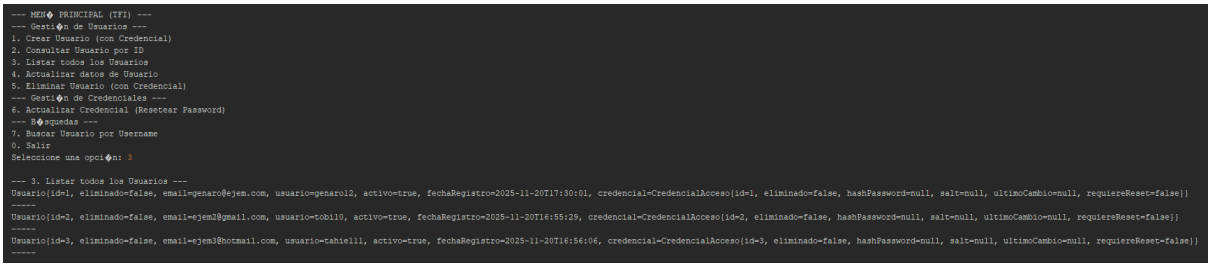
### Reglas de negocio:

- Un usuario eliminado no puede autenticarse.
- Si una credencial tiene *"requiere_reset = TRUE"*, el sistema puede obligarlo a actualizar la clave.
- No se permite actualizar un usuario inexistente.
- No se permite asignar una credencial a usuario que ya tiene una.

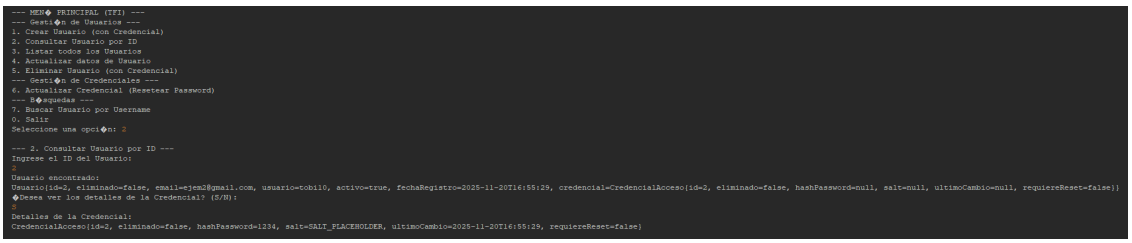
## 8. Pruebas realizadas:

### Pruebas manuales de consola:

Prueba de consola listando los usuarios:




Acá buscamos un usuario por su ID y muestramos su credencial:



## Pruebas de consultas útiles SQL:

### Verificación general de base de datos:

```
1 • SELECT * FROM usuario; -- Consultar todos los usuarios, incluidos los eliminados.
2 • SELECT * FROM credencial_acceso; -- Consultar todas las credenciales incluidas las eliminadas.
3
```

Result Grid

| id | eliminado | hash_password | salt | ultimo_cambio | requiere_reset | usuario_id |
|---|---|---|---|---|---|---|
| 4 | 0 | 5f4dcc3b5aa765d61d8327deb882cf99 | abc123 | 2024-01-15 10:30:00 | 0 | 1 |
| 5 | 0 | a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6 | xyz789 | 2024-03-10 09:15:00 | 1 | 3 |
| 6 | 1 | 098f6bcd4621d373cade4e832627b4f6 | oldsalt | 2023-12-01 11:00:00 | 1 | 4 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

Result Grid

| id | eliminado | username | email | activo | fecha_registro |
|---|---|---|---|---|---|
| 1 | 0 | ADMIN | admin@sistema.com | 1 | 2024-01-15 10:30:00 |
| 2 | 0 | JPEREZ | jperez@mail.com | 1 | 2024-02-20 14:00:00 |
| 3 | 0 | MGARCIA | mgarcia@mail.com | 0 | 2024-03-10 09:15:00 |
| 4 | 1 | VELIMINADO | eliminado@mail.com | 0 | 2023-12-01 11:00:00 |

### Validación de la relación 1 → 1:

```
4   -- Validar la relacion 1-->1
5 • SELECT u.id, u.username, c.id AS cred_id, c.hash_password
6   FROM usuario u
7   LEFT JOIN credencial_acceso c ON u.id = c.usuario_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| id | username | cred_id | hash_password |
|---|---|---|---|
| 1 | ADMIN | 4 | 5f4dcc3b5aa765d61d8327deb882cf99 |
| 2 | JPEREZ | NULL | NULL |
| 3 | MGARCIA | 5 | a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6 |
| 4 | VELIMINADO | 6 | 098f6bcd4621d373cade4e832627b4f6 |

### Detección de inconsistencias en la relación:

```
9   -- Encontrar credenciales sin usuarios
10 • SELECT *
11  FROM credencial_acceso
12  WHERE usuario_id IS NULL;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

| id | eliminado | hash_password | salt | ultimo_cambio | requiere_reset | usuario_id |
|---|---|---|---|---|---|---|
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

```sql
-- Encontrar usuarios sin credenciales
SELECT *
FROM usuario
WHERE id NOT IN (SELECT usuario_id FROM credencial_acceso WHERE usuario_id IS NOT NULL);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| id | eliminado | username | email | activo | fecha_registro |
|---|---|---|---|---|---|
| 2 | 0 | JPEREZ | jperez@mail.com | 1 | 2024-02-20 14:00:00 |
| NULL | NULL | NULL | NULL | NULL | NULL |

```sql
-- Verificar que la FK funcione correctamente
SELECT usuario_id
FROM credencial_acceso
WHERE usuario_id NOT IN (SELECT id FROM usuario);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| usuario_id |
|---|

**Validación del uso de baja lógica:**

```sql
-- Validar baja lógica
SELECT *
FROM usuario
WHERE eliminado = FALSE;
```

Result Grid | Filter Rows: | Edit: | Export/Impor

| id | eliminado | username | email | activo | fecha_registro |
|---|---|---|---|---|---|
| 1 | 0 | ADMIN | admin@sistema.com | 1 | 2024-01-15 10:30:00 |
| 2 | 0 | JPEREZ | jperez@mail.com | 1 | 2024-02-20 14:00:00 |
| 3 | 0 | MGARCIA | mgarcia@mail.com | 0 | 2024-03-10 09:15:00 |
| NULL | NULL | NULL | NULL | NULL | NULL |

```
29    -- Ver credenciales eliminadas
30 •  SELECT *
31    FROM credencial_acceso
32    WHERE eliminado = TRUE;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| id | eliminado | hash_password | salt | ultimo_cambio | requiere_reset | usuario_id |
|---|---|---|---|---|---|---|
| 6 | 1 | 098f6bcd4621d373cade4e832627b4f6 | oldsalt | 2023-12-01 11:00:00 | 1 | 4 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

**Consulta útil o frecuente en el sistema:**

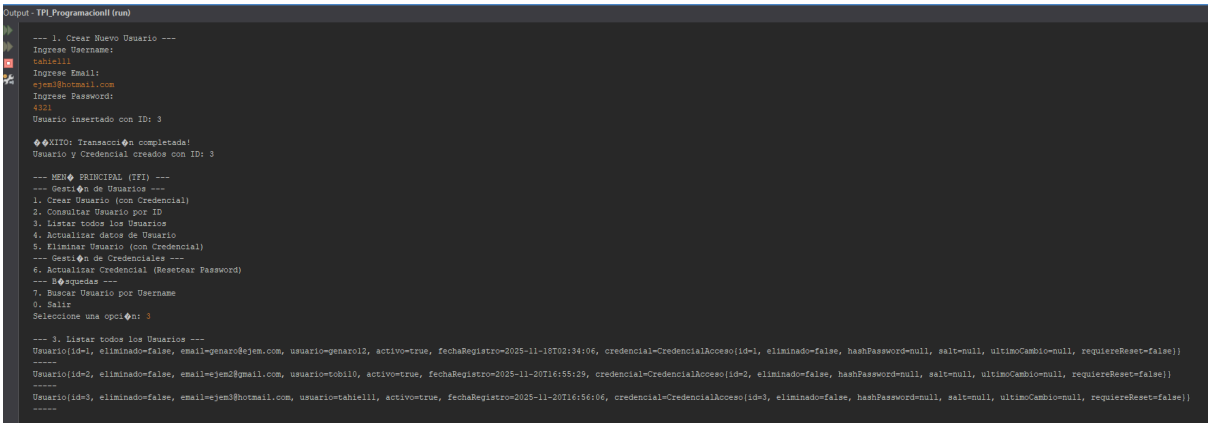

```
29    -- Ver credenciales eliminadas
30 •  SELECT *
31    FROM credencial_acceso
32    WHERE eliminado = TRUE;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

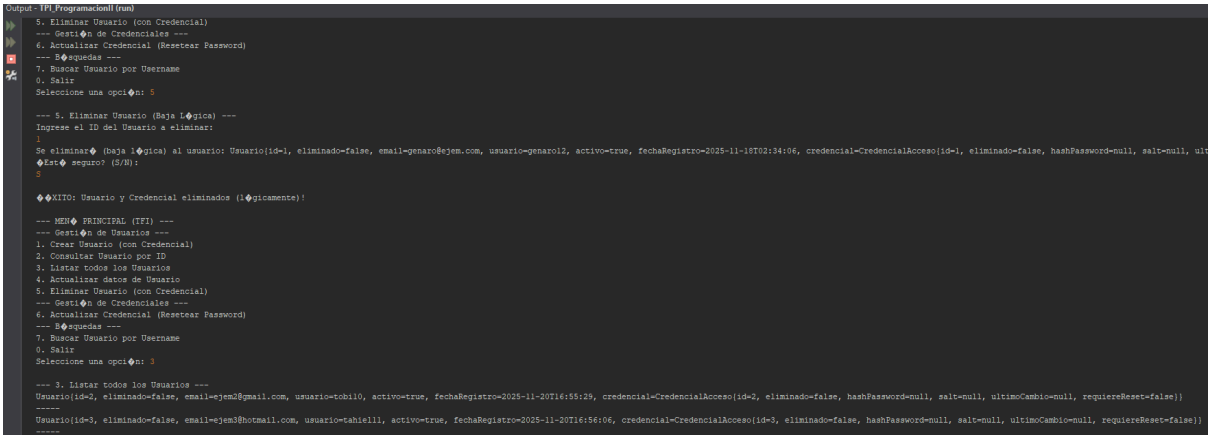
| id | eliminado | hash_password | salt | ultimo_cambio | requiere_reset | usuario_id |
|---|---|---|---|---|---|---|
| 6 | 1 | 098f6bcd4621d373cade4e832627b4f6 | oldsalt | 2023-12-01 11:00:00 | 1 | 4 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |


**Pruebas de transacciones:**

En esta primera prueba, se pueden ver 3 usuarios guardados en la base de datos y más arriba de la misma consola, como se pudo crear y guardar correctamente un usuario generando a su vez, una credencial para cada uno de forma automatica:



En la siguiente captura podemos ver cómo hacemos un eliminado lógico y mostramos nuevamente la lista para observar que no figura el usuario eliminado y se da de baja también la credencial al mismo:




## 9. Conclusiones y mejoras futuras:

### Conclusiones:

Durante este TFI se ha logrado con éxito diseñar e implementar una aplicación en Java que cumple con los requisitos planteados:

- Definición del Dominio.
- Creación de Base de datos con lenguaje SQL, con tablas “usuario” y “CredencialAcceso”, aplicando baja lógica mediante “eliminado”, utilizando restricciones “UNIQUE” en “username”, “email” y “usuario_id” y se aplico una clave foránea (FK) para asegurar la integridad referencial.
- Se estructuró el código en capas, como la configuración de la base de datos (DatabaseConecction), entidades, capa DAO genérica y específica, capa de servicio de transacciones (commit y rollback) y lógica de negocio.
- Se implementaron todas las operaciones CRUD mediante JDBC y el patrón DAO.
- El menú de consola permite la interacción del usuario, y las validaciones realizadas.
- Creación de diagrama UML de clases y decisiones de diseño.

En resumen este TFI nos permitió simular todas las etapas para una creación de una aplicación mediante java con conexión a la base de datos, utilizando un menú interactivo para el usuario y a su vez experimentamos la etapa de creación y diseño prematuro mediante la creación del diagrama UML

### Mejoras futuras:

Como futuras mejoras a este programa se podría implementar:

- Mayor seguridad en las credenciales; utilizando algoritmos de hashing, incorporando salt aleatorias e intentos fallidos para el ingreso con una clave.
- Evolución hacia una interfaz web; reemplazando el menú de consola a una interfaz web.
- Implementación de roles y permisos; agregando un rol diferenciando a los usuarios y dependiendo del nivel filtrar sus niveles de acceso.
- Paginación y filtrados de búsqueda; mejorando los listados por filtrados de email, estado, fecha de creación, etc.
- Auditoría de cambios realizados; registrar las modificaciones realizadas a usuarios y credenciales, permitiendo una mejor auditoría y control.
- Pruebas automatizadas; para reforzar la estabilidad y funcionamiento de transacciones, validaciones y excepciones.

## 10. Citar fuentes y herramientas utilizadas:

**Fuentes externas utilizadas:**

En esta ocasión el grupo logró resolver las dudas presentadas a la hora de realizar el TFI mediante las teorías y videos otorgados por la institución.

**Herramientas utilizadas:**

- Java 24.
- MySQL Workbench.
- Dbeaver y phpMyAdmin.
- UMLetino.
- Apache NetBeans.

**Link del video subido a YT:** https://youtu.be/-Ch3IKkJYCU