TRABAJO FINAL INTEGRADOR

(Informe técnico)

ALUMNA:

ALBRIGI MARIANELA
ESTEBAN BOHORQUEZ
BONANNO NICOLÁS
NOWELL JULIETA

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN
UNIVERSIDAD TECNOLÓGICA NACIONAL

PROGRAMACIÓN II

Docente Titular
Ariel Enferrel

Docente Tutor
Federico Frankerberger

18 de NOVIEMBRE de 2025

INTEGRANTES Y ROLES

- **Integrante A:** **Diseño y Configuración del Proyecto**

Se encargó del setup inicial del sistema y de toda la configuración técnica base.
Organizó la estructura de paquetes (config, entities, dao, service, main) y creó el archivo db.properties con los datos necesarios para conectarse a la base.
Desarrolló la clase DatabaseConnection, capaz de leer propiedades externas y entregar una conexión estable y segura, manejando correctamente excepciones de I/O y SQL.
Además, colaboró en el diseño UML del sistema, definiendo los paquetes principales y las clases Empleado y Legajo con su relación uno a uno.

- **Integrante B:** **Entidades y DAO Layer**

Se ocupó del modelo de datos (entities) y de implementar toda la capa de persistencia.
Diseñó las clases Empleado y Legajo con sus atributos, métodos y representaciones legibles.
Definió la interfaz GenericDAO y creó las implementaciones EmpleadoDAO y LegajoDAO, utilizando siempre PreparedStatement y aceptando conexiones externas para facilitar las transacciones.
Opcionalmente realizó pruebas básicas de los CRUD para verificar que la interacción con la base funcionara correctamente.

- **Integrante C:** **Service Layer y Reglas de Negocio**

Desarrolló la lógica de negocio y el manejo de transacciones.
Implementó servicios genéricos y concretos para Empleado y Legajo, aplicando reglas como: validar campos obligatorios, evitar múltiples legajos por empleado y realizar baja lógica modificando el flag eliminado.
Se encargó de coordinar transacciones completas usando una única conexión (commit/rollback) y de capturar errores, registrarlos y traducirlos a mensajes de negocio más claros para la capa superior.

- **Integrante D:** **Interfaz de Usuario y Control de Flujo**

Realizó el menú de consola y toda la interacción con el usuario final.
Programó la clase AppMenu, con menús y submenús para operaciones CRUD de Empleado y Legajo.
Gestionó las entradas del usuario con validaciones de formato, manejo de excepciones y mensajes claros.
Conectó el menú con los servicios para ejecutar operaciones reales y probó el flujo completo, incluyendo escenarios con commit y rollback para asegurar un comportamiento correcto en la ejecución real.

ELECCIÓN DEL DOMINIO Y JUSTIFICACIÓN

- **Dominio elegido:** ***EMPLEADO - LEGAJO***

El grupo eligió el dominio Empleado – Legajo porque representa un caso realista, sencillo y aplicable a múltiples entornos organizacionales. Este tipo de relación es común en sistemas de gestión de personal, donde cada empleado cuenta con un legajo único que centraliza su

información administrativa y laboral. Además, este dominio permite aplicar conceptos clave de modelado de datos y programación orientada a objetos, como:

- Relaciones uno a uno (1→1) entre entidades.
- Validaciones y restricciones (campos únicos, obligatorios, longitudes máximas).
- Buenas prácticas de diseño, como el uso de baja lógica y enumeraciones.

DECISIONES RELATIVAS AL DISEÑO DE LA BASE DE DATOS Y LA ARQUITECTURA

DISEÑO DE LA BASE DE DATOS

En el diseño de la base de datos, la relación entre Empleado y Legajo se implementó como *1 a 1* mediante una clave foránea única (legajo_id) en Empleado apuntando al campo id de la tabla Legajo, garantizando que cada empleado tenga exactamente un legajo y cada legajo pertenezca a un único empleado. Aunque era posible usar una PK compartida, se optó por la FK única porque permite que Legajo exista de manera independiente de Empleado durante procesos internos, simplifica la generación de IDs al ser Empleado autoincremental y facilita la gestión de transacciones al insertar primero el empleado y luego su legajo. Esta decisión asegura la integridad referencial, evita duplicados o legajos huérfanos y prioriza flexibilidad y manejo seguro de operaciones transaccionales. Además, los atributos críticos, como nroLegajo, dni y estado, se definieron como NOT NULL y con restricciones de unicidad donde corresponde, permitiendo realizar altas, bajas lógicas y actualizaciones de manera controlada y consistente.

DISEÑO DE LA ARQUITECTURA

La aplicación sigue una arquitectura por capas que separa claramente las responsabilidades de cada componente. La capa de entidades modela los objetos de negocio y sus atributos, la capa DAO gestiona la persistencia y puede recibir conexiones externas para permitir transacciones coordinadas, la capa Service centraliza la lógica de negocio y el control de transacciones, y la capa de aplicación o menú actúa como interfaz con el usuario sin involucrarse en la lógica de negocio ni la persistencia. Esta separación facilita la mantenibilidad, la escalabilidad y las pruebas, garantizando que cada capa cumpla un rol específico dentro del sistema.

A continuación, se detallan las responsabilidades y funcionalidades de cada una de las capas que integran el sistema:

PAQUETE CONFIG

Este paquete pertenece a la capa de Componentes Transversales o Configuración.
Responsabilidad: Su principal responsabilidad es aislar los parámetros de configuración sensibles y específicos del entorno (como credenciales de bases de datos) del código fuente. Proporciona los valores necesarios para que la clase DatabaseConnection pueda establecer y gestionar la conexión con la base de datos subyacente.

PAQUETE DE PERSISTENCIA (DAO)

Este paquete se encarga exclusivamente de la persistencia y de aislar al resto del sistema de los detalles de la base de datos. Usa los objetos Empleado y Legajo del paquete Dominio para interactuar con la BD.

Responsabilidades principales:

- **CRUD:** Gestiona crear, leer, actualizar y eliminar registros en las tablas empleado y legajo.
- **Mapeo Objeto-Relacional:** Convierte objetos Java a SQL y viceversa (mapResultSetToEmpleado, mapResultSetToLegajo).
- **Gestión de Conexiones:** Abre y cierra conexiones a la BD de forma segura (DataBaseConnection.getConnection()).
- **Transacciones:** Permite operaciones independientes autónomos (manejan su propia conexión/transacción) y transaccionales (reciben una conexión externa), permitiendo agrupar múltiples operaciones DAO en una sola transacción lógica.

PAQUETE ENTITIES

Este paquete representa la capa más interna y fundamental de la aplicación. Su única responsabilidad es modelar la información y las reglas estructurales del negocio. No debe tener dependencias de la capa DAO, Servicio o Presentación.

Responsabilidades principales:

- **Representación del Negocio:** Contener las clases que representan los conceptos clave del dominio (Entidades: Empleado, Legajo).
- **Encapsulamiento de Datos:** Proporcionar los atributos y los métodos *getter* y *setter* para gestionar el estado interno de las entidades.
- **Definición de Identidad y Relaciones:** Definir las claves primarias (id en Base) y las relaciones entre entidades (la asociación 1:1 de Empleado con Legajo).
- **Lógica de Objeto Intrínseca:** Implementar lógica básica de objetos como la igualdad (equals) y el *hashing* (hashCode), basándose en atributos clave del negocio (DNI y número de Legajo).
- **Reutilización de Propiedades Base:** Utilizar herencia para centralizar atributos comunes a todas las entidades persistentes (como el ID y el indicador de baja lógica).

PAQUETE MAIN

Este paquete cumple la función de la Capa de Presentación para un programa de consola. Se encarga de manejar la interacción con el usuario y coordinar el flujo general de la aplicación.

Responsabilidades principales:

- **Inicialización y Composición del Sistema:** La clase AppMenu actúa como punto de arranque y crea la cadena de dependencias del sistema. Allí se instancian manualmente los DAOs (EmpleadoDAO, LegajoDAO), los servicios (EmpleadoService, LegajoService) y el controlador (MenuController). Esto garantiza que cada capa reciba los componentes que necesita para funcionar correctamente.
- **Gestión del Flujo de la Aplicación:** AppMenu controla la ejecución completa del programa. Muestra el menú principal, mantiene el ciclo de ejecución hasta que el usuario elige salir e interpreta la opción ingresada para delegarla al controlador correspondiente.

- **Interacción con el Usuario (I/O):** La clase MenuController administra todas las entradas del usuario, captura los valores ingresados y realiza validaciones básicas como IDs, DNIs o estados. También maneja errores comunes sin interrumpir la ejecución del sistema y delega todas las operaciones reales a los servicios de negocio.
- **Utilidades de Presentación y Entorno:** El paquete incluye clases auxiliares como MenuDisplay, que muestra el menú principal sin lógica adicional, y TestConnection, utilizada al iniciar el desarrollo para confirmar que la base de datos estuviera disponible antes de ejecutar la aplicación.

PAQUETE SERVICE

Este paquete centraliza la lógica de negocio. Sus componentes actúan como intermediarios entre la capa de persistencia y la capa de presentación.

Responsabilidades principales:

La capa service aplica toda la lógica de negocio antes de llegar a la base de datos. Esto incluye validar que el DNI sea único, que tenga el formato correcto, que los campos obligatorios estén completos y que el email cumpla con el patrón esperado. En el caso de EmpleadoService también se encarga de crear automáticamente el legajo asociado, generando su número, asignando valores por defecto y asegurando la relación uno a uno.

También administra las transacciones. Abre una transacción desactivando el auto-commit, ejecuta todas las operaciones necesarias y, si todo sale bien, hace el commit. Si algo falla, ejecuta un rollback para dejar la base en un estado consistente. Al final siempre cierra la conexión y restaura el auto-commit.

La capa service no accede directamente a la base; coordina a los DAOs. Recibe los DAOs por constructor y combina sus métodos para resolver las operaciones de negocio. Por ejemplo, un insert de empleado puede llamar al insert transaccional del empleado, al insert del legajo y luego a una actualización para vincularlos, todo dentro de una misma operación lógica.

Finalmente maneja los errores de negocio. Convierte fallas de validación o problemas del DAO en excepciones claras para la capa superior. Puede lanzar IllegalArgumentException cuando la validación falla o excepciones más generales cuando hay problemas internos. Además registra en el logger los pasos importantes, incluyendo errores, commits y rollbacks, para ayudar en el diagnóstico.

PAQUETE UTILS

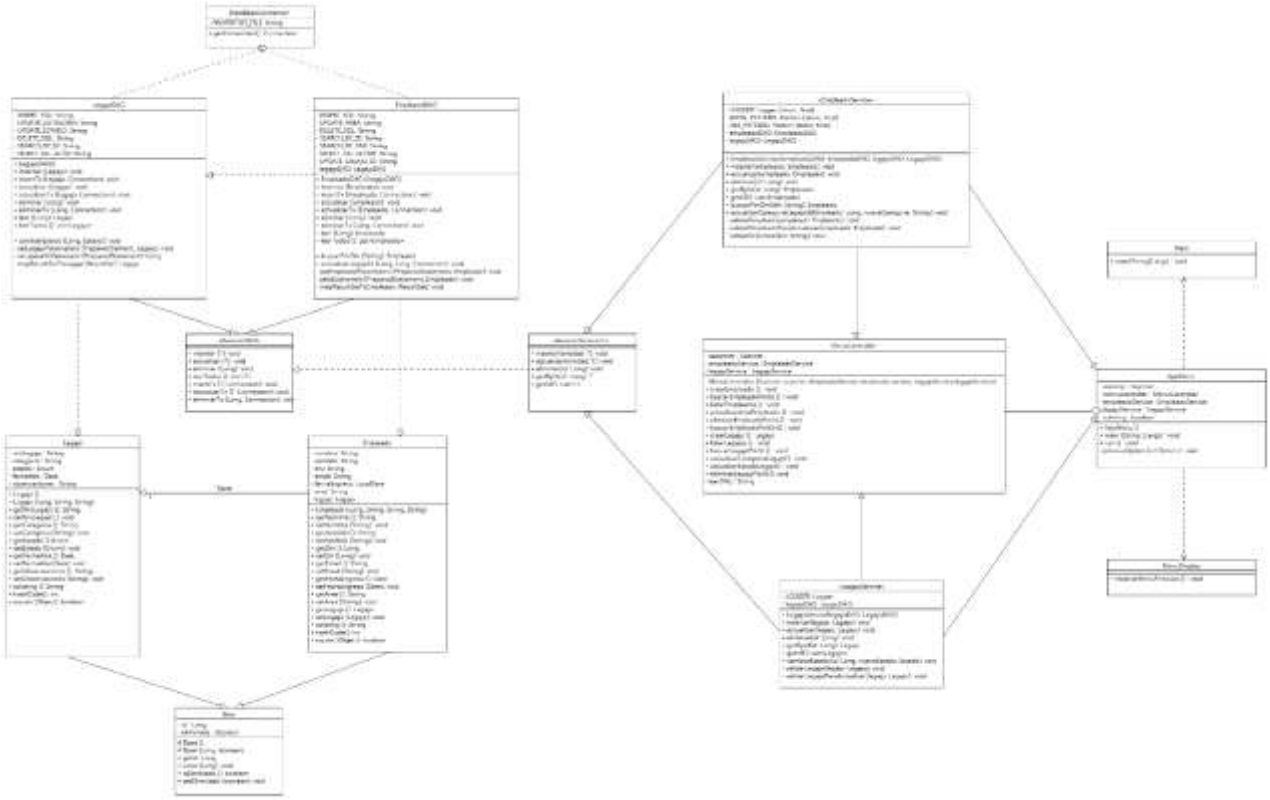
La responsabilidad de la clase DataBaseConnection dentro del paquete utils se centra exclusivamente en el suministro de conexiones a la base de datos, encapsulando la configuración de la infraestructura.

Se encarga de localizar y leer el archivo de configuración externo especificado por la constante PROPERTIES_FILE. Esto desacopla las credenciales y detalles de la conexión del código fuente.

Encapsulamiento de la Configuración: Carga la URL, el host, el puerto, el nombre de la base de datos, el usuario y la contraseña de manera segura desde el archivo de propiedades.

Construye la URL completa del protocolo JDBC utilizando los parámetros leídos del archivo de propiedades. Utiliza el método estático DriverManager.getConnection() para establecer una

nueva conexión física con la base de datos MySQL en cada llamada al método estático getConnection().
La clase está diseñada como una utilidad estática, lo que significa que otras capas (como service y dao) simplemente la llaman para obtener una conexión sin preocuparse por la implementación subyacente.
Maneja las excepciones relacionadas con la lectura del archivo (IOException) y los problemas de conexión (que son propagados como SQLException), asegurando que cualquier fallo en la infraestructura se maneje adecuadamente. Los errores de lectura se encapsulan en un RuntimeException para detener la aplicación si la configuración inicial es defectuosa.



(Para una correcta visualización, se adjunta el archivo UML en el repositorio)

PERSISTENCIA Y TRANSACCIONES

En este proyecto, la persistencia de los datos se garantiza tanto desde la base de datos como desde el sistema Java, para que la información siempre esté segura y consistente.

Persistencia desde la base de datos

La base de datos mantiene la integridad de los datos gracias a:

- Relaciones bien definidas, como la relación 1:1 entre *Empleado* y *Legajo*, donde la tabla empleado tiene la clave foránea legajo_id que apunta a la PK id de legajo.
- Restricciones de integridad como claves primarias, NOT NULL y valores válidos para campos como estado o categoría.
- Autogeneración de IDs, evitando duplicados y conflictos.

- Baja lógica, en lugar de eliminar registros físicamente, lo que protege la información ante errores o borrados accidentales.

Con estas reglas, la base de datos asegura que los datos sean coherentes incluso si ocurre un fallo en el sistema.

Estructura de la base de datos

Tabla: empleado
id (PK, autoincremental)
nombre (NOT NULL)
apellido (NOT NULL)
dni (NOT NULL, UNIQUE)
email
fecha_ingreso
area
legajo_id (FK → legajo.id)
eliminado (baja lógica)

Tabla: legajo
id (PK, autoincremental)
nro_legajo (NOT NULL, UNIQUE)
categoria
estado (NOT NULL)
fecha_alta
observaciones
eliminado (baja lógica)

Persistencia desde el sistema Java

En el sistema, la persistencia se asegura mediante:

- La relación 1:1 definida en la base de datos, en el sistema Java se implementa en la clase Empleado, la cual contiene un atributo de tipo Legajo. Así, cada instancia de empleado tiene asociado su legajo correspondiente.

- DAO especializados, que centralizan todas las operaciones de acceso a la base (CRUD), evitando errores y duplicación de código.
- Uso de PreparedStatement, lo que garantiza que los datos se envíen correctamente y previene ataques de inyección SQL.
- Mapeo consistente entre BD y objetos Java, mediante métodos que convierten los ResultSet en objetos Empleado o Legajo, asegurando que lo que está en la base se represente correctamente en el sistema.
- Control de transacciones, usando commit y rollback en las operaciones críticas. Por ejemplo, cuando se inserta un empleado y su legajo asociado:
    - Si ambas transacciones son exitosas, entonces se ejecuta commit y los cambios se guardan en la BD.
    - Si algo falla en cualquier paso, entonces se ejecuta rollback y la base vuelve al estado anterior, evitando inconsistencias (como un empleado sin legajo o un legajo huérfano).

FLUJO DE PERSISTENCIA: INTERACCIÓN ENTRE DAO Y SERVICE

En nuestro sistema, todas las operaciones sobre los datos pasan por un flujo bien definido que involucra al Service y al DAO, para mantener la persistencia y consistencia entre la base de datos y los objetos Java. La idea es que el Service maneje las validaciones y la lógica de negocio, y el DAO centralice todas las consultas y modificaciones a la base de datos.

OPERACIONES QUE MODIFICAN DATOS (insertar, actualizar, eliminar, actualizar categoría o estado):

1. **Validación en el Service:** se verifican campos obligatorios, formatos correctos (como DNI o email) y reglas de negocio (por ejemplo, DNI único, existencia de legajo). Si la validación falla, se lanza una excepción y no se realizan cambios en la base de datos.
2. **Apertura de conexión y control de transacción:** se obtiene una conexión a la base y se desactiva el autoCommit. Esto permite agrupar varias operaciones (por ejemplo, insertar un empleado y su legajo) en una sola transacción atómica.
3. **Ejecución de operaciones en el DAO:** se usan métodos especializados (insertTx, actualizarTx, eliminarTx) que reciben la conexión activa. Se realizan los cambios en la base de datos usando PreparedStatement para garantizar seguridad y consistencia.
4. **Commit o rollback:** si todas las operaciones se ejecutan correctamente, se hace commit y los cambios quedan guardados en la base. Si ocurre cualquier error, se hace rollback y la base vuelve al estado anterior, evitando inconsistencias como un empleado sin legajo o un legajo huérfano.
5. **Cierre de conexión**: se vuelve a activar autoCommit y se cierra la conexión para liberar recursos.

OPERACIONES DE LECTURA (leer por ID, listar todos, buscar por DNI o ID):

1. **Validación en el Service:** se verifica que los parámetros sean válidos (ID mayor a 0, DNI con el formato correcto, etc.).
2. **Consulta en el DAO:** se ejecuta un query con PreparedStatement y se mapea el ResultSet a objetos Java (Empleado o Legajo).
3. **Devolución del resultado:** el Service retorna el objeto o la lista al resto del sistema.
4. Como no se modifica nada en la base, no se necesita commit ni rollback, pero el flujo sigue pasando por el DAO para mantener la consistencia y centralizar el acceso a la base.

En resumen, la persistencia del sistema está compuesta por la base de datos, su modelo relacional y la implementación en los DAO, que en conjunto permiten realizar operaciones confiables, consistentes y claras sobre los datos.

VALIDACIONES Y REGLAS DE NEGOCIO

En el sistema, antes de interactuar con la base de datos, se realizan varias **validaciones** para garantizar que los datos sean correctos, coherentes y cumplan las reglas del dominio:

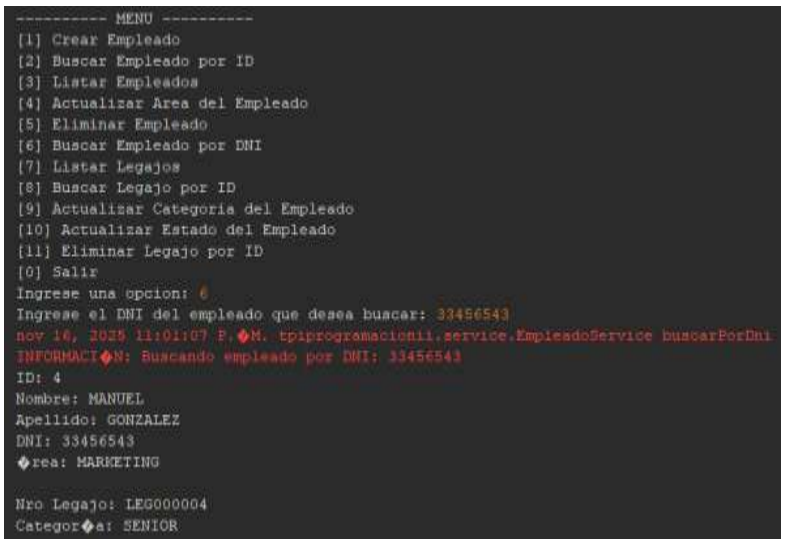
1. **Campos obligatorios y no nulos**
   - Para un **Empleado**, los campos nombre, apellido y dni son obligatorios. Además, cuando se actualiza un empleado, el área también debe estar presente.
   - Para un **Legajo**, los campos nroLegajo, categoria y estado no pueden ser nulos. Esto asegura que cada registro tenga la información mínima necesaria para ser válido.

2. **Restricciones de formato y dominio**
   - nombre y apellido deben tener al menos 2 caracteres.
   - área también debe tener mínimo 2 caracteres.
   - dni debe ser un número de 7 u 8 dígitos, evitando entradas inválidas.
   - email (si se ingresa) debe respetar un formato válido de correo.
   - Los valores de estado en Legajo solo pueden ser ACTIVO o INACTIVO.
   - La categoria de Legajo no puede estar vacía y se normaliza a mayúsculas al actualizar.

3. **Reglas de negocio**
   - **DNI único:** no puede existir más de un empleado con el mismo DNI, garantizando identificación única.
   - **Relación 1:1 entre Empleado y Legajo:** cada empleado tiene exactamente un legajo, evitando registros huérfanos.
   - **Baja lógica:** al eliminar un empleado o legajo, no se borra físicamente la información; se marca como eliminado para mantener la integridad histórica.
   - **Transacciones atómicas:** operaciones críticas como insertar un empleado y su legajo asociado se realizan dentro de una transacción. Si algo falla, se hace rollback y los datos vuelven al estado anterior, evitando inconsistencias.

Estas validaciones y reglas de negocio aseguran que los datos en la base y en el sistema siempre sean coherentes, completos y respeten las restricciones del dominio, evitando errores y protegiendo la información.

PRUEBAS DE MENU: ***crear empleado con legajo asociado.***



PRUEBAS DE MENU: *buscar empleado por dni.*

```
---------- MENU ----------
[1] Crear Empleado
[2] Buscar Empleado por ID
[3] Listar Empleados
[4] Actualizar Area del Empleado
[5] Eliminar Empleado
[6] Buscar Empleado por DNI
[7] Listar Legajos
[8] Buscar Legajo por ID
[9] Actualizar Categoria del Empleado
[10] Actualizar Estado del Empleado
[11] Eliminar Legajo por ID
[0] Salir
Ingrese una opcion: 6
Ingrese el DNI del empleado que desea buscar: 33456543
nov 16, 2025 11:01:07 P.�M. tpiprogramacionii.service.EmpleadoService buscarPorDni
INFORMACI�N: Buscando empleado por DNI: 33456543
ID: 4
Nombre: MANUEL
Apellido: GONZALEZ
DNI: 33456543
�rea: MARKETING

Nro Legajo: LEG000004
Categor�a: SENIOR
```

PRUEBAS Y CONSULTAS SQL

- **INSERT de legajo**

```
-- ------------------------------
-- Datos de la tabla legajo
-- ------------------------------
INSERT INTO legajo (nro_legajo, categoria, estado, fecha_alta, observaciones)
VALUES
('LEG001001', 'Administrativo', 'ACTIVO', '2022-03-10', 'Ingreso por concurso');

SELECT * FROM legajo where id = 1001
```

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

id	eliminado	nro_legajo	categoria	estado	fecha_alta	observacion...
1001	0	LEG001001	Junior	ACTIVO	2019-04-19	

- **INSERT de empleado (asignando el legajo anteriormente creado)**

```
-- ------------------------------
-- Datos de la tabla empleado
-- ------------------------------
INSERT INTO empleado (nombre, apellido, dni, email, fecha_ingreso, area, legajo_id)
VALUES
('Juan', 'Pérez', '30123456', 'juan.perez@example.com', '2022-03-10', 'Administración', 1001);

SELECT * FROM empleado WHERE dni = '30123456';
```

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

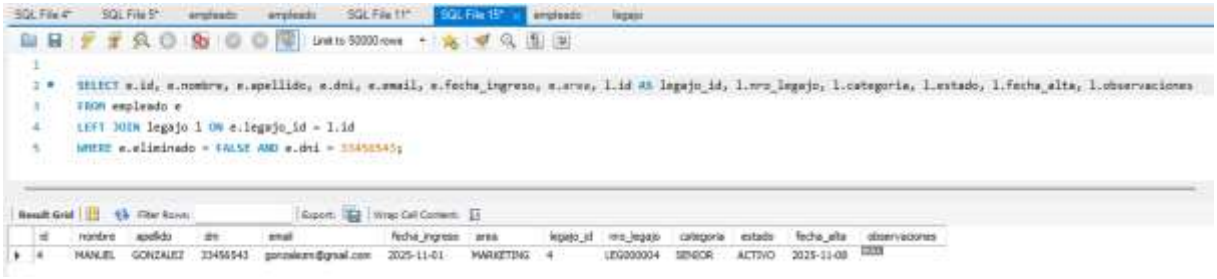
id	eliminado	nombre	apellido	dni	email	fecha_ingreso	area	legajo_id
933885	0	Juan	Pérez	30123456	juan.perez@example.com	2022-03-10	Administración	1001

empleado 12

Action Output

	Time	Action	Response
1	12:54:42	INSERT INTO empleado (nombre, apellido, dni, email, fech...	1 row(s) affected
2	12:54:45	SELECT * FROM empleado WHERE dni = '30123456' LIMI...	1 row(s) returned

- **BÚSQUEDA EMPLEADO POR DNI**



CONCLUSIONES Y MEJORAS

El desarrollo de este proyecto permitió aplicar de manera práctica una arquitectura por capas, integrando conceptos de entidades, acceso a datos, servicios y presentación. Se construyó una aplicación funcional capaz de gestionar empleados y sus legajos, incorporando operaciones completas de creación, búsqueda, actualización y eliminación, junto con manejo transaccional para asegurar la integridad de los datos.
Además del aspecto técnico, uno de los puntos más importantes del trabajo fue la dinámica del grupo. El avance del proyecto dependió de una comunicación constante, tanto para resolver problemas como para coordinar criterios de diseño y funcionamiento. Cada decisión requirió retroalimentación y acuerdos dentro del equipo, lo que generó un ida y vuelta permanente y enriquecedor.
Este proceso colaborativo fue esencial para mantener el proyecto alineado y avanzar de manera ordenada. La experiencia reflejó exactamente el objetivo académico de este tipo de trabajos: aprender a desarrollarse en equipo, compartir responsabilidades, apoyarse mutuamente y sostener un proyecto común desde el inicio hasta su finalización.

Respecto de las mejoras, se podría realizar las siguientes implementaciones:

Interfaz del usuario: Implementar una interfaz grafica, para brindar una mejor experiencia al usuario.
Gestión de usuarios y permisos: Incorporar un modulo de usuarios y roles, esto permitiría restringir quien puede crear, eliminar o modificar empleados y legajos.
Búsquedas y filtros más avanzados: Podrían agregarse filtros por área, categoría del legajo, estado (ACTIVO/INACTIVO) o rangos de fecha de ingreso. Esto ampliaría la capacidad de consulta del sistema y lo volvería más flexible para distintos tipos de análisis y necesidades operativas.

FUENTES Y HERRAMIENTAS UTILIZADAS:

Para la realización del trabajo se utilizaron los contenidos proporcionados por la materia y se emplearon herramientas de inteligencia artificial como apoyo en su elaboración.