Tecnicatura Universitaria en Programación a Distancia

Programación II

Coord. Carlos Martinez
Prof. Ariel Enferrel (Comisión 3 - Tutor Tomás Ferro)
Prof. Cinthia Rigoni (Comisión 10 - Tutor Francisco Quarñolo)
Prof. Alberto Cortez (Comisión 14 - Tutor Ramiro Hualpa)

Trabajo Final Integrador

(Grupo 10: Empleado - Legajo)

Alumnos:

Agustin Emiliano Sotelo Carmelich (agustinemiliano22@gmail.com) - Comisión 10

Bruno Giuliano Vapore (brunogvapore@gmail.com) - Comisión 10

Santiago Octavio Varela (santiago.varela@tupad.utn.edu.ar) - Comisión 14

Diego Alejandro Velardes (velardesdiego@gmail.com) - Comisión 3

Fecha de Entrega: 17 de Noviembre de 2025

Índice

Justificación de la elección del dominio

Se seleccionó el dominio Empleado (A) → Legajo (B) por su aplicabilidad para cumplir con todos los objetivos técnicos y de negocio establecidos en el Trabajo Final Integrador. Para comenzar con la justificación, en primer lugar, el requisito más importante de este trabajo es modelar una asociación unidireccional 1 a 1. El dominio Empleado → Legajo representa este escenario de forma esperable y común en cualquier sistema de gestión de personal, ya que un Empleado posee generalmente un único Legajo en su expediente personal y, por lo tanto, un Legajo pertenece a un único Empleado

Esta correlación directa facilita la implementación de la restricción 1 a 1 en la base de datos, ya sea mediante una clave foránea única (UNIQUE FOREIGN KEY) en la tabla de Legajo apuntando a Empleado o usando una clave primaria compartida (PK/FK), ambas opciones permitidas por la consigna.

También el proyecto exige la ejecución de operaciones compuestas mediante transacciones (commit/rollback). El caso de uso ideal para demostrar este concepto es el alta de un Empleado (entidad A), ya que requiere la creación simultánea de su Legajo (entidad B), sumado a que ambas operaciones (INSERT en empleados e INSERT en legajos) deben ejecutarse como una unidad indivisible (atomicidad). Si el INSERT del Empleado es exitoso, pero el INSERT del Legajo falla (por ejemplo, por violar la restricción UNIQUE del nro_legajo), la transacción completa debe revertirse (rollback). El Service gestionará esta lógica, asegurando que no existan empleados sin su correspondiente legajo.

Otro motivo importante para la elección del dominio elegido es que la unidireccionalidad es clara, un Empleado hace referencia a Legajo. Esto se alinea con el flujo de negocio más habitual, que implica que la consulta operativa principal es "dado un empleado, obtener su legajo" y no a la inversa.

Como último aspecto a considerar en la justificación del dominio, es su robustez para establecer Reglas de Negocio y Validación claras. El dominio Empleado → Legajo proporciona un conjunto interesante de campos que permiten implementar validaciones significativas en la capa de Service y búsquedas específicas en el menú. Allí se deben gestionar restricciones UNIQUE en campos críticos como dni y email del Empleado, así como con número del Legajo, lo que entraría en validaciones de unicidad. También se deben establecer validaciones de formato, como ocurre con el campo email mencionado, que requerirá una validación de formato. Además, las validaciones deben contener manejo de tipos de datos, como ocurrirá con java.time.LocalDate (fechaIngreso, fechaAlta) y Enum (estado en Legajo), cumpliendo con la especificación. Por último, las búsquedas por campo relevante también son parte de la validación y los métodos de búsqueda son utilizados por los validadores para verificar las reglas de negocio.

Validaciones y Reglas de Negocio

El sistema de gestión de personal se fundamenta en un conjunto de reglas de negocio diseñadas para garantizar la integridad y coherencia de la información. La correcta identificación del personal es lo más importante, por lo que cada empleado debe registrarse con nombre, apellido y DNI, que son campos definidos como obligatorios para asegurar un registro de datos mínimo y consistente. El DNI funciona como identificador único e

irrepetible en toda la organización. Se contempla un campo para el email que, si bien es opcional, debe cumplir con un formato estándar y se define también como único para prevenir conflictos de comunicación y acceso al sistema.

Para reflejar con precisión el ciclo de vida del empleado, el sistema distingue entre dos conceptos clave. Primero, el estado del legajo (ACTIVO o INACTIVO), que define la situación contractual de la persona. Segundo, una bandera de eliminado a nivel de empleado, un mecanismo de borrado lógico reservado para gestionar registros creados por error, ocultándose de la operativa diaria sin perder el dato en caso de que se realice una auditoría. Del mismo modo, se diferencian dos fechas fundamentales: la fechaIngreso del empleado, que marca el inicio de su relación contractual y la fechaAlta del legajo, que corresponde a su creación administrativa en el sistema. Esta separación permite modelar con fidelidad el proceso real, donde puede existir un desfase entre ambas fechas, siendo por lo general la primera que se agrega la de la entidad empleado.

Finalmente, se establece una estricta relación uno a uno entre un empleado y su legajo, asegurando que no haya duplicidades. Esta cardinalidad se implementa de forma robusta a nivel físico mediante una restricción UNIQUE sobre la clave foránea del empleado en la tabla de legajos. Esta medida impide que un mismo empleado pueda ser asociado a más de un legajo, garantizando la unicidad de la relación. La integridad se refuerza con una política de borrado en cascada, la cual subraya que un legajo no tiene entidad propia y depende completamente del ciclo de vida de su empleado asociado, automatizando así la consistencia del sistema y evitando datos huérfanos. (Ver Anexo I: Diagrama Entidad-Relación).

Diseño

Decisiones clave y diagrama UML

Decisiones clave en la persistencia 1 a 1

Para modelar la relación unidireccional 1 a 1 entre Empleado y Legajo, tomamos una decisión estratégica sobre cómo representar esta relación en la base de datos MySQL. Esta fue utilizar una Clave Foránea Única (FK Única), donde la tabla legajo (B) tiene su propia clave primaria (id) autoincremental, con una columna separada (empleado_id) que actúa como clave foránea con restricción UNIQUE.

Es posible fundamentar esta elección en base a varios puntos. En primer lugar, al usar esta estrategia la entidad Legajo mantiene una identidad única e independiente del Empleado. Suponiendo que el sistema escale en el futuro y se necesitara referenciar a un Legajo, se podría hacer fácilmente. Un segundo punto es el carácter explícito del modelo elegido, dado que se diferencian fácilmente la PK (id) y la FK (empleado_id) en la tabla Legajo, cuestión que no ocurriría con una clave compuesta ya que una columna cumple dos roles a la vez. Un tercer ítem es que esta forma garantiza la unicidad mediante la restricción UNIQUE sobre la columna empleado_id en la tabla legajo, que habilita a que Empleado no pueda tener más de un Legajo. Como última cuestión, este modelo también es mucho más claro en cuanto a cómo se alinea con DAO, dado que EmpleadoDAO crea un Empleado obteniendo su id, que Legajo recibe mediante empleado_id (mediante service, como veremos) para guardarlo en la columna FK. (Ver Anexo II: Diagrama UML).

Arquitectura por capas (responsabilidades de cada paquete)

El objetivo principal de la arquitectura por capas, siendo uno de los aspectos más importantes de este proyecto, es estructurar el código de manera ordenada, mantenible y escalable. Al separar la aplicación en paquetes con responsabilidades únicas (como config, entities, dao, service y main), logramos concretar algunos de los principios clave de la Programación Orientada a Objetos. Uno de ellos es la alta cohesión, ya que cada paquete se enfoca en una única responsabilidad) y, por otro lado, el bajo acoplamiento, debido a que los paquetes son lo más independientes posible entre sí. Esta separación clara es esencial para gestionar la complejidad del sistema y facilitar las pruebas unitarias en cada caso.

Basándonos en las consignas y en nuestro dominio elegido, la arquitectura y sus responsabilidades se definen de la siguiente manera:

Paquete config/

La responsabilidad de este paquete es gestionar la conexión a la base de datos MySQL. Por un lado, `DatabaseConnetion` es la clase que centraliza el acceso a la base de datos, siendo su principal tarea proveer un método que retorne un nuevo objeto `java.sql.Connection`; es también el responsable de leer las credenciales (URL, usuario, password) para establecer la conexión con MySQL.

Luego está `TransactionManager`, que es una clase auxiliar que contiene una `Connection`. Su propósito es encapsular el ciclo de vida en una transacción. Justamente, la capa de servicio usará esta clase para iniciar la transacción, hacer `commit()` o `rollback()` según sea necesario y pasar esa conexión gestionada a la capa DAO. Al finalizar, asegura que la conexión se cierre y se restaure a autoCommit(true) de forma segura.

Paquete entities/

El paquete `entities` tiene como principal responsabilidad el modelado de las clases del dominio elegido. Las clases que contiene son Base, Empleado y Legajo, junto con un enum que simplemente define los valores permitidos para el atributo estado de la clase Legajo, llamado EstadoLegajo, asegurando la integridad de ese dato.

Respecto a las clases, Base es una clase abstracta que actúa como una plantilla para las otras dos. Define los atributos comunes `id` y `eliminado`, así como sus métodos de acceso, proveyendo la base para la baja lógica requerida en todo el sistema, siguiendo las consignas.

Por otro lado, Empleado (Clase A) representa al empleado, heredando de Base lo ya mencionado y conteniendo atributos propios, como `nombre`, `apellido` y `dni`, `email`, `fechaIngreso` y `area`. También contiene la referencia unidireccional 1 a 1 solicitada con el atributo `private Legajo legajo`.

Por último, Legajo (Clase B) es la restante, que representa legajo y también hereda de Base. Contiene atributos propios como `numeroLegajo`, `categoria`, `estado`, `fechaAlta` y `observaciones`, y no tiene una referencia de vuelta a Empleado, respetando el tipo de asociación mencionada.

Paquete dao/

La responsabilidad principal de la capa DAO es la comunicación con la base de datos SQL. Dada esta tarea, también es la encargada de la persistencia de los datos a través del conjunto de acciones representado conceptualmente por CRUD. Dentro de este paquete, la interfaz GenericDAO es la que define el contrato CRUD estándar mediante sus

métodos (crear, leer, actualizar, eliminar). Además define métodos transaccionales (crearTx, actualizarTx, eliminarTx) que aceptan una `Connection` externa. Esto es lo que permite a la capa service gestionar operaciones complejas dentro de una única transacción.

Como clases, tenemos EmpleadoDAO (en referencia a la clase A) y LegajoDAO (clase B), ambas implementando el contrato establecido en GenericDAO. La primera tiene la responsabilidad de materializar la relación entre ambas entidades en la base de datos a través de los métodos de lectura, mapeo y escritura mencionados. Los métodos de lectura (`leer` y `leerTodos`, para SELECT) usan LEFT JOIN para traer los datos del Empleado y su Legajo en una sola consulta. Por el lado del mapeo, el método `mapResultSetToEmpleado` es el encargado de "llenar" el objeto Empleado con el resultado de las consultas a la base de datos (los datos) que se ejecuten y también, si Legajo existe, retoma los datos del legajo gracias a la verificación que hace con LEFT JOIN. Los métodos de escritura de EmpleadoDAO, en tercer lugar, se encargan exclusivamente de insertar, actualizar o eliminar datos de la tabla Empleado, delegando a la capa service la coordinación con LegajoDAO y el manejo de la tabla Legajo.

LegajoDAO es un poco más simple por la unidireccionalidad de la relación entre la clase A y la B. Su responsabilidad más importante es manejar los datos que le vienen a través de la clave foránea `empleado_id`, ya establecida en la clase Legajo y gestionada por la capa service, que configura y asigna ese nuevo ID en el objeto Legajo. En otras palabras, LegajoDAO hace persistente en la base de datos el ID que ya está dentro del objeto Legajo.

Como último aspecto, aplicamos las buenas prácticas sugeridas y vistas en los materiales, como lo son el uso de `PreparedStatement` en todas las operaciones para prevenir SQL Injection; el manejo de la baja lógica, no física; el uso de métodos auxiliares dentro de las clases, con la finalidad de no repetir código dentro de los métodos generales implementados (sin Tx y con Tx).

Paquete service/

Este paquete tiene la responsabilidad de orquestar las operaciones, aplicar las reglas de negocio (validaciones) y gestionar las transacciones. Sus contenidos clave son la interfaz GenericService y las clases LegajoServiceImpl y EmpleadoServiceImpl, que reciben la interfaz.

GenericService es la interfaz que estandariza las operaciones de negocio (`insertar`, `eliminar`, `actualizar`, `getById`, `getAll`). Esto asegura que la capa main pueda interactuar con cualquier servicio (EmpleadoService o LegajoService) de una manera consistente, cumpliendo con los requisitos solicitados para este trabajo.

Respecto a la clase LegajoServiceImpl, esta se encarga de aplicar las reglas de negocio para Legajo antes de llamar al paquete DAO. Esto puede observarse, por ejemplo, en el método `validateLegajo`, donde se comprueba que los datos no sean nulos, no estén vacíos y que respeten las longitudes máximas establecidas. En getById o GetAll, simplemente se delega la llamada a DAO, ya que no se requiere una lógica de negocio adicional.

En tercer lugar, EmpleadoServiceImpl se encarga de coordinar operaciones entre Empleado y Legajo, validando tanto las reglas para Empleado con métodos internos como llamando a LegajoServiceImpl para que valide el Legajo. Además es el encargado de iniciar transacciones con TransactionManager (paquete config/), llamar a EmpleadoDAO y LegajoDAO para realizar cambios en los datos y, como punto importante, asignar la FK (el

ID) al objeto legajo. En caso de que las llamadas a ambas clases DAO funcionen sin excepciones, EmpleadoServiceImpl se encarga de hacer tm.commit() y, por el contrario, si hay una falla (en validaciones o por SQLException del DAO), el catch va a llamar a tm.rollback()

Paquete main/

La capa Main es la de presentación y la de interfaz de usuario (UI) de este proyecto. Es el único punto de contacto con el usuario final de la aplicación. Su responsabilidad es mostrar información y recopilar datos de entrada a través de la consola. Esta capa no contiene lógica de negocio (validaciones complejas) ni lógica de persistencia (SQL). Este paquete está compuesto por las clases Main, AppMenu, MenuDisplay y MenuHandler.

La clase Main es el punto de entrada único de la aplicación. Su principal responsabilidad será lanzar el menú, como lo indica la consigna en el trabajo, contiene una cantidad de líneas de código mínima respecto a otras clases.

AppMenu es la clase que organiza centralmente la aplicación y ensambla las capas; en el método `createEmpleadoService()` justamente une toda las partes de la aplicación, creando instancias en los DAOs (EmpleadoDAO, LegajoDAO) y luego inyectándolas en los servicios (LegajoServiceImpl, EmpleadoServiceImpl). Sumado a ello, es parte importante del ciclo de vida de la aplicación en su totalidad, ya que gestiona el bucle principal `run()` que mantiene corriendo la aplicación dado que ejecuta un `while(running)`.

Respecto a la parte del menú, MenuHandler es el controlador de las operaciones del menú, ejecutando la lógica de cada opción dentro de él. Recibe las dependencias desde AppMenu (Scanner y EmpleadoServiceImpl), se encarga de interactuar con el usuario haciéndole preguntas específicas para ese fin y una vez recopila los datos, llama al método correspondiente de la capa service para comenzar con las operaciones. También es responsable de capturar las excepciones que vienen de service y mostrárselas al usuario de una forma más amigable. MenuDisplay, por su parte, solo es una clase para mostrar el texto del menú, teniendo como principal método `mostrarMenuPrincipal()` que imprime las opciones del menú en la consola.

Persistencia

Estructura de la base de datos (Relación 1-a-1)

Para modelar la relación unidireccional 1 a 1 entre Empleado y Legajo, se implementó la estrategia de Clave Foránea Única, una de las opciones recomendadas por la consigna. En esta estrategia, la tabla empleado (A) es la entidad principal, que contiene su clave primaria y los campos requeridos en función de la descripción del dominio sugerida (nombre, apellido, dni, etc). La tabla legajo (B) es la entidad dependiente de A y además de contener su propia clave primaria, contiene la columna de `empleado_id`. Respecto a la FOREING KEY (FK) en la tabla legajo, esta apunta a empleado(id), asegurando la integridad referencial, en donde un Legajo no puede existir sin un Empleado asociado.

A su vez, esta estructura está presente en los DAOs, ya que EmpleadoDAO se enfoca en la tabla empleado y Legajo DAO, en la de legajo, siendo este último en escribir empleado_id (FK)

Orden de las operaciones

La arquitectura DAO ofrece un orden estricto para mantener la integridad referencial al crear o leer la relación establecida, cuestión que buscamos seguir en nuestro caso. Además del orden y gestión de la lectura y las transacciones, también decidimos mencionar el orden de creación como otro aspecto importante en este punto.

Orden de Creación

Para crear un Empleado con su Legajo de forma atómica, EmpleadoServiceImpl ordena las operaciones de la siguiente manera, siendo el encargado de creación:

1. Se inicia la transacción en la capa Service. Primero se llama a`empleadoDAO.crearTx()`, por lo que la base de datos genera el id de nuevo Empleado. El EmpleadoDAO recupera este id y lo devuelve a Service, actualizando el objeto Empleado en memoria.
2. El Service toma este id y lo asigna al objeto Legajo. Se llama a `legajoDao.crearTx()`, que persiste el Legajo con la empleado_id (FK) correcta, vinculando el Legajo al Empleado. Si ambos son exitosos, Service confirma la transacción.

Orden de Lectura

La lectura de la relación es responsabilidad exclusiva del EmpleadoDAO. Primero se ejecuta una consulta SELECT que utiliza LEFT JOIN para unir la tabla empleado (A) con legajo (B). Luego el método `mapResultSetToEmpleado` mapea los datos de `ResultSet` y asignarlos al objeto Empleado con los valores correspondientes, mientras que si a través del JOIN trae un id para legajo válido, además crea el objeto Legajo y lo asigna usando `empleado.setLegajo()`.

Gestion de Transacciones (Commit / Rollback)

El control transaccional (commit/rollback) es responsabilidad exclusiva de la capa Service, por lo que las DAO son transaccionalmente pasivas ya que solo ejecutan las consultas SQL usando la Connection que reciben a través de los métodos `Tx()` y teniendo a la capa Config que provee la posibilidad de realizar esa conexión.

La razón fundamental de todo esto es que Service es la única capa que entiende la unidad de trabajo completa desde la perspectiva del negocio, coordinando a los DAOs para cumplir las reglas de negocio, validándolas. La transacción comienza cuando la capa de servicio obtiene una conexión de DatabaseConnection y, fundamentalmente, deshabilita el modo autocommit llamando a `conn.setAutoCommit(false)`. Este es el momento exacto en que la transacción JDBC comienza.

El commit() se invoca explícitamente `conn.commit()` al final del bloque try. Esto solo ocurre si todas las operaciones de la unidad de trabajo (las llamadas a `Tx`) se completaron sin lanzar ninguna excepción. El commit hace permanentes todos los cambios de la transacción. Respecto al rollback, este se invoca con `conn.rollback()` dentro del bloque catch. Si cualquier operación dentro del try falla (ya sea una validación de negocio o una SQLException del DAO), la ejecución salta al catch, el commit nunca se ejecuta y el rollback revierte todos los cambios que se hayan realizado dentro de la transacción. Finalmente, en el bloque finally, la conexión se restaura a su estado normal con `conn.setAutoCommit(true)` y se cierra. Esto asegura que la conexión se devuelva correctamente, independientemente de si la transacción fue exitosa o no.

Pruebas Realizadas

Captura del Menú por consola:

```
========= MENU PRINCIPAL =========
--- Gestión de Empleados ---
1. Crear Empleado (y Legajo)
2. Listar Empleados
3. Actualizar Empleado
4. Eliminar Empleado
5. Buscar Empleado por ID
6. Buscar Empleado por DNI
---- Gestión de Legajos ----
7. Listar Legajos
8. Listar Legajos por Estado
------------------------------
0. Salir
Ingrese una opcion:
```

Consultas SQL:

Proceso de creación de Empleado y Legajo (Opción 1) ; y búsqueda por DNI (Opción 6).

```
Ingrese una opcion: 1
== Crear Empleado y Legajo ==
Nombre: Tomas
Apellido: Duplech
DNI: 30471256
Email: tomasD@gmail.com
Área: Ventas
Fecha de Ingreso (AAAA-MM-DD): 2025-11-01

--- Datos del Legajo (Requerido) ---
Número de Legajo: VTS-2025-001
Seleccione el estado del legajo:
  1 - ACTIVO
  2 - INACTIVO
Opción: 1
Categoría: Junior
Observaciones (opcional): Documentacion pendiente
Fecha de Alta (AAAA-MM-DD): 2025-11-17

¡Empleado y legajo creados exitosamente!

Presione Enter para continuar...
```

```
Ingrese una opcion: 6
== Buscar empleado por DNI ==
DNI del empleado a buscar: 30471256

Empleado encontrado:
Empleado (ID: 22)
Nombre: Tomas
Apellido: Duplech
DNI: 30471256
Email: tomasD@gmail.com
Fecha de Ingreso: 2025-11-01
Area: Ventas
Eliminado: false
Legajo: Legajo [Número: VTS-2025-001, Categoría: Junior, Estado: ACTIVO, Fecha Alta: 2025-11-17, Observaciones: Documentacion pendiente]

Presione Enter para continuar...
```

Se pueden observar otras consultas SQL útiles realizadas. (Ver Anexo III).

Conclusiones

Para concluir este trabajo, en búsqueda del cumplimiento de los requisitos exigidos, podemos afirmar que hemos consolidado el aprendizaje sobre cómo se debe construir software robusto y mantenible desde nuestro lugar como estudiantes. El punto más importante de nuestro diseño ha sido la separación de responsabilidades, algo que es un supuesto para usar una arquitectura por capas en el desarrollo de software. Esta separación se materializó en una separación por paquetes en Java, que representa cada una de las capas, lo más definida posible (main, service, dao, entities y config).

El mayor desafío estuvo presente en la conexión con la base de datos y cómo service y la capa dao operaban sobre este proceso, ya que es uno de los últimos temas vistos durante la cursada. Entendimos la importancia de la capa service como gestor y organizador de la lógica de negocio, además de la validación de las reglas, y su relación con DAO, coordinando la implementación de las transacciones que DAO hace persistente en la base de datos. En cuanto a la estrategia, la elección de la Clave Foránea Única se demostró, como vimos, acertada, ya que nos otorgó la garantía de integridad 1 a 1 necesaria para cumplir con la consigna y simplificar la lógica de nuestros servicios.

En vista a futuras implementaciones, se podrían agregar varios elementos. Podríamos automatizar la inyección de dependencias usando un framework de Inversión de Control (IoC), así como podríamos exponer nuestra lógica de negocio (service) como una API REST. Por último, teniendo lo realizado como base, podríamos reemplazar la interfaz de consola por una interfaz gráfica (GUI) con JavaFX o Swing (menos usado en la actualidad), demostrando una clara separación de capas al no tener que modificar la lógica de negocio o de datos.

Herramientas Utilizadas

DBeaver Corporation. (s.f.). DBeaver Community [Software]. https://dbeaver.io/

GitHub, Inc. (s.f.). GitHub [Servicio web]. https://github.com/

Google. (2025). Gemini [Software de inteligencia artificial]. https://gemini.google.com/

Homebrew. (s.f.). Homebrew [Software]. https://brew.sh/

Oracle Corporation. (s.f.-a). Java SE Development Kit [Software]. https://www.oracle.com/java/

Oracle Corporation. (s.f.-b). MySQL Community Server [Software]. https://www.mysql.com/

Software Freedom Conservancy. (s.f.). Git [Software]. https://git-scm.com/

The Apache Software Foundation. (s.f.). Apache NetBeans [Software]. https://netbeans.apache.org/

Bibliografía

Bloch, J. (2018). Effective Java (3ª ed.). Addison-Wesley Professional.

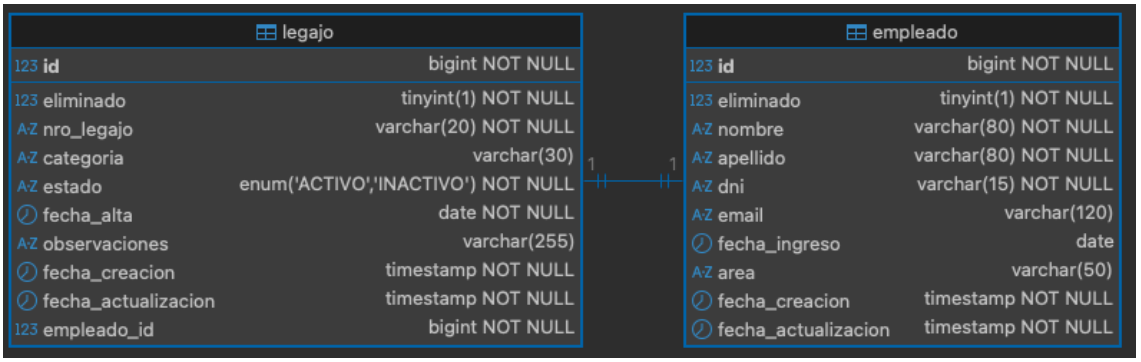
Fowler, M. (2002). Patterns of enterprise application architecture. Addison-Wesley Professional.

Oracle. (s.f.-a). Lesson: JDBC basics. The Java™ Tutorials. Consultado el 12 de noviembre de 2025, de https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

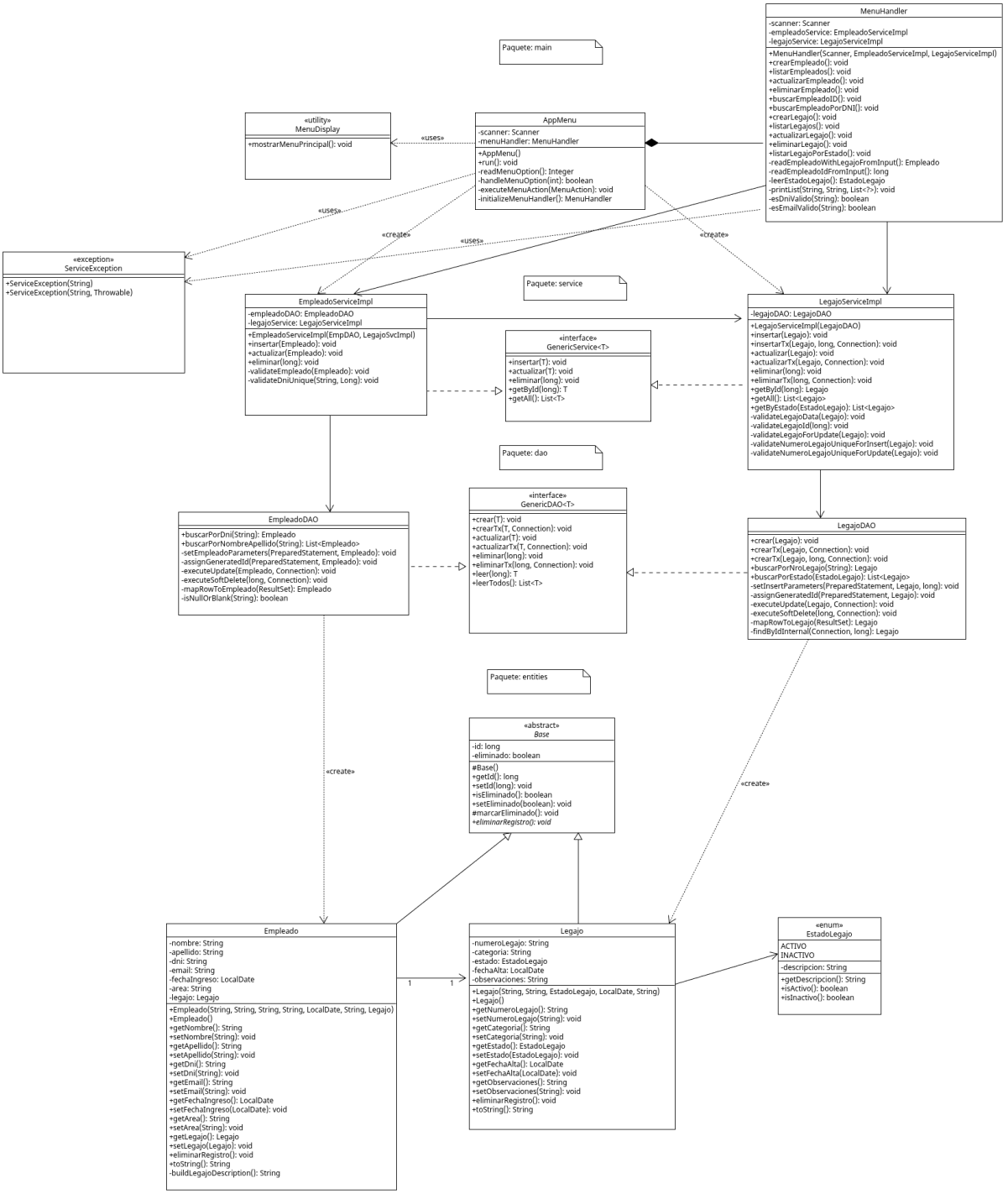
Oracle. (s.f.-b). MySQL 8.0 reference manual. Oracle Help Center. Consultado el 12 de noviembre de 2025, de https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Database system concepts (7ª ed.). McGraw-Hill.

Anexo I: Diagrama Entidad-Relación



Anexo II: Diagrama UML



Anexo III: Otras Consultas SQL

Opción 2: Listando empleados.

```
------------------------------
Empleado (ID: 15)
Nombre: Javier
Apellido: Rios
DNI: 33665544
Email: javier.rios@empresa.com
Fecha de Ingreso: 2021-05-19
Area: Soporte
Eliminado: false
Legajo: No asignado
------------------------------
Empleado (ID: 19)
Nombre: Gabriela
Apellido: Molina
DNI: 33889900
Email: gabriela.molina@empresa.com
Fecha de Ingreso: 2020-09-10
Area: Ventas
Eliminado: false
Legajo: Legajo [Número: VTS-2020-002, Categoría: Semi-Senior, Estado: INACTIVO, Fecha Alta: 2020-09-10, Observaciones: Licencia médica prolongada]
------------------------------
Empleado (ID: 20)
Nombre: Fernando
Apellido: Luna
DNI: 36667788
Email: fernando.luna@empresa.com
Fecha de Ingreso: 2024-01-08
Area: Marketing
Eliminado: false
Legajo: Legajo [Número: MKT-2024-001, Categoría: Trainee, Estado: ACTIVO, Fecha Alta: 2024-01-08, Observaciones: null]
------------------------------
Empleado (ID: 22)
Nombre: Tomas
Apellido: Duplech
DNI: 30471256
Email: tomasD@gmail.com
Fecha de Ingreso: 2025-11-01
Area: Ventas
Eliminado: false
Legajo: Legajo [Número: VTS-2025-001, Categoría: Junior, Estado: ACTIVO, Fecha Alta: 2025-11-17, Observaciones: Documentacion pendiente]
------------------------------
```

Opción 3: Actualizando datos (sin y con errores).

```
Ingrese una opcion: 3
== Actualizar Empleado ==
ID del empleado a buscar: 22

Editando datos del empleado. Deje el campo en blanco para no modificar.
Nombre [Tomas]:
Apellido [Duplech]: Duplex
DNI [30471256]:
Email [tomasD@gmail.com]: todasD@hotmail.com
Área [Ventas]:
Fecha de Ingreso (AAAA-MM-DD) [2025-11-01]:

--- Editando datos del Legajo ---
Número de Legajo [VTS-2025-001]:
Categoría [Junior]:
Estado del legajo [ACTIVO]:
  1 - ACTIVO
  2 - INACTIVO
Opción (deje en blanco para no cambiar): 2
Observaciones [Documentacion pendiente]: Renuncia. Liquidación Pendiente.
Fecha de Alta (AAAA-MM-DD) [2025-11-17]:

¡Empleado actualizado exitosamente!

Presione Enter para continuar...
```

```
ID del empleado a buscar: 15

[ERROR DE DATOS]: Error de datos: El empleado no tiene un legajo asociado para actualizar.

Presione Enter para continuar...
```

```
Ingrese una opcion: 3
== Actualizar Empleado ==
ID del empleado a buscar: 20

Editando datos del empleado. Deje el campo en blanco para no modificar.
Nombre [Fernando]:
Apellido [Luna]:
DNI [36667788]:
Email [fernando.luna@empresa.com]: fernando.luna.empresa.com
Área [Marketing]:
Fecha de Ingreso (AAAA-MM-DD) [2024-01-08]:

--- Editando datos del Legajo ---
Número de Legajo [MKT-2024-001]:
Categoría [Trainee]:
Estado del legajo [ACTIVO]:
  1 - ACTIVO
  2 - INACTIVO
Opción (deje en blanco para no cambiar):
Observaciones [null]:
Fecha de Alta (AAAA-MM-DD) [2024-01-08]:


Presione Enter para continuar...
[ERROR DE DATOS]: Email inv�lido. Formato esperado: usuario@dominio.com.
```

Opción 8: Listando legajos por algún estado (Inactivo).

```
Ingrese una opcion: 8
== Listar Legajos por Estado ==
Seleccione el estado del legajo:
  1 - ACTIVO
  2 - INACTIVO
Opción: 2
--- Listando Legajos con estado INACTIVO ---
Legajo [Número: VTS-2020-002, Categoría: Semi-Senior, Estado: INACTIVO, Fecha Alta: 2020-09-10, Observaciones: Licencia médica prolongada]
----------------------------
Legajo [Número: VTS-2025-001, Categoría: Junior, Estado: INACTIVO, Fecha Alta: 2025-11-17, Observaciones: Renuncia. Liquidación Pendiente.]
----------------------------

Presione Enter para continuar...
```

Opción 4: Eliminando un empleado por su ID.

```
Ingrese una opcion: 4
== Eliminar Empleado ==
ID del empleado a buscar: 22

Empleado con ID 22 eliminado exitosamente (junto con su legajo).

Presione Enter para continuar...
```

Opción 5: Buscando un empleado por su ID.

```
Ingrese una opcion: 5
== Buscar empleado por ID ==
ID del empleado a buscar: 22


[ERROR DE DATOS]: No se encontr  ning n empleado con el ID: 22
Presione Enter para continuar...
```

```
Ingrese una opcion: 5
== Buscar empleado por ID ==
ID del empleado a buscar: 15
Empleado encontrado:
Empleado (ID: 15)
Nombre: Javier
Apellido: Ríos
DNI: 33665544
Email: javier.rios@empresa.com
Fecha de Ingreso: 2021-05-19
Area: Soporte
Eliminado: false
Legajo: No asignado

Presione Enter para continuar...
```

Opción 7: Listando Legajos (todos).

```
Ingrese una opcion: 7
--- Listando Legajos ---
Legajo [Número: VTS-2020-002, Categoría: Semi-Senior, Estado: INACTIVO, Fecha Alta: 2020-09-10, Observaciones: Licencia médica prolongada]
------------------------------
Legajo [Número: MKT-2024-001, Categoría: Trainee, Estado: ACTIVO, Fecha Alta: 2024-01-08, Observaciones: null]
------------------------------

Presione Enter para continuar...
```

Anexo IV: Links a repositorio GitHub y Video

Repositorio: https://github.com/enkai12/TFI_Programacion2_Grupo10

Video: https://youtu.be/tlba6bTBGB0

Video (link secundario): https://drive.google.com/file/d/1tR6CAjF2fkM8lgNX4Pv6hndnKGzz3X3H/view?usp=sharing