TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

UTN
UNIVERSIDAD TECNOLÓGICA NACIONAL

Trabajo Final Integrador

Programación II

Grupo 147: Vehiculo → SeguroVehicular

Grela, Ariel **(Comisión 3)**
Higa, Matías **(Comisión 6)**

Roles x Integrante

Rol	Grela, Ariel	Higa, Matías
UML y Arquitectura		🟢
SQL		🟢
Capa entities	🟢	🟢
Capa config		🟢
Capa DAO	🟢	
Capa service (validaciones y transacciones)	🟢	
AppMenu y pruebas	🟢	
Documentación (informe + README)		🟢
Revisión (formateo y ajustes)	🟢	🟢
Video	🟢	🟢

Dominio

Vehiculo → SeguroVehicular

Clase Vehiculo (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
dominio	String	NOT NULL, **UNIQUE**, máx. 10
marca	String	NOT NULL, máx. 50
modelo	String	NOT NULL, máx. 50
anio	Integer	
nroChasis	String	**UNIQUE**, máx. 50
seguro	SeguroVehicular	Referencia 1→1 a B

Clase SeguroVehicular (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
aseguradora	String	NOT NULL, máx. 80
nroPoliza	String	**UNIQUE**, máx. 50
cobertura	Enum {RC, TERCEROS, TODO_RIESGO}	NOT NULL
vencimiento	java.time.LocalDate	NOT NULL

Para el desarrollo de ambos trabajos (Programación II y BD1) elegimos el dominio *Vehículo -> SeguroVehicular* porque es una temática que nos interesa y entendemos que puede representar un caso real de uso en el cual aplicar los contenidos vistos en las materias.

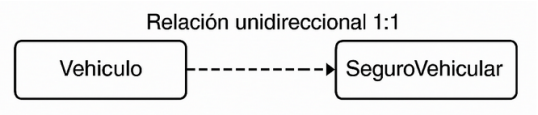
Diseño + UML

Sobre el Diseño y la BD

El sistema se diseñó siguiendo una arquitectura en capas, buscando separar la lógica de negocio, la persistencia, la interacción mediante consola y la configuración de la BD.

Se diseñó una relación unidireccional 1:1 desde Vehiculo hacia SeguroVehicular, alineando así el trabajo con la lógica del mundo real:

- Cada vehículo tiene una única póliza de seguro asociada.
- Cada póliza corresponde a un vehículo específico.



Vehículo (A) contiene una referencia a SeguroVehicular (B), definido en una relación de composición.

En relación a la base de datos, optamos por mantener la base hecha para Base de Datos I, con solo algunas modificaciones. Para la tabla Vehiculo de la base de datos:

- La relación se asegura mediante el campo *vehiculo.seguro_id*, con una restricción *UNIQUE* que impide asignar un mismo seguro a más de un vehículo.
- *FOREIGN KEY (seguro_id) REFERENCES segurovehicular(id)* refleja el sentido conceptual de la relación: un vehículo “posee” un seguro. ON DELETE CASCADE define la composición.

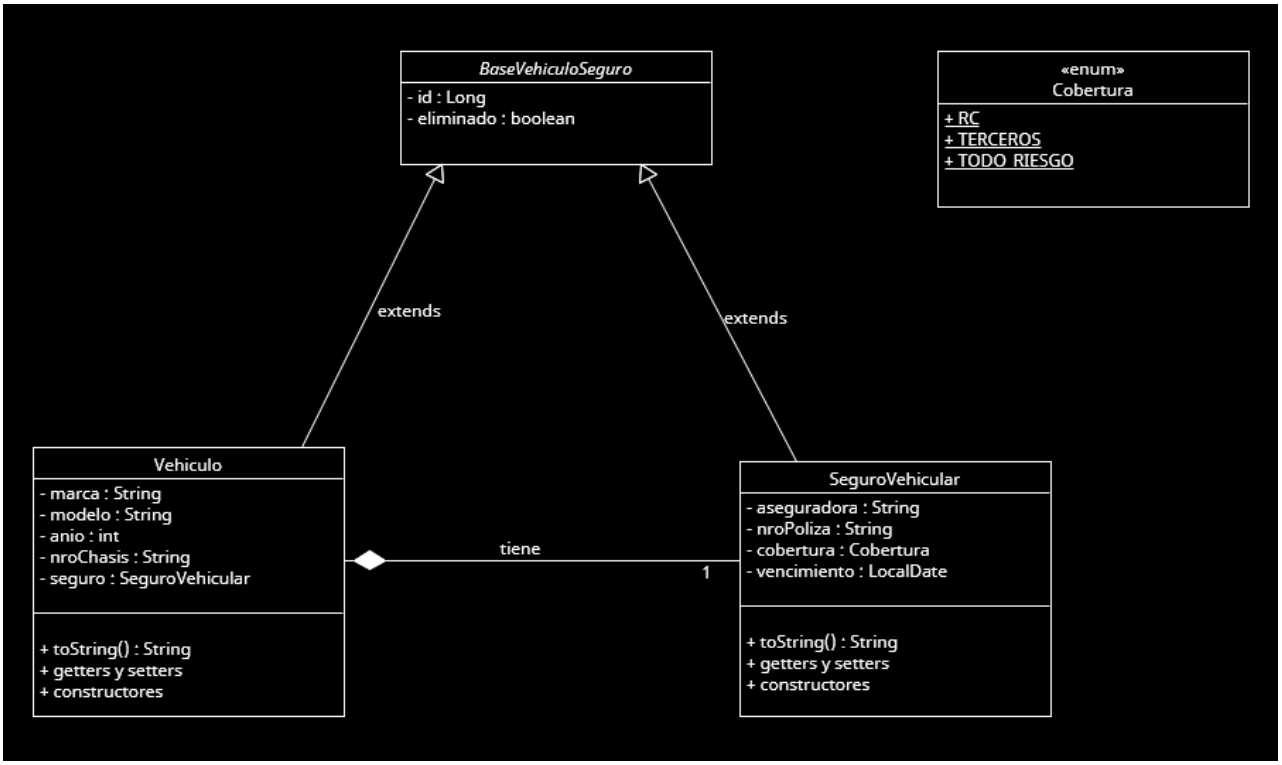
```
17 • CREATE TABLE Vehiculo (
18       id BIGINT AUTO_INCREMENT PRIMARY KEY NOT NULL,
19       eliminado BOOLEAN,
20       dominio VARCHAR(10) NOT NULL UNIQUE,
21       marca VARCHAR(50) NOT NULL,
22       modelo VARCHAR(50) NOT NULL,
23       anio INT(4),
24       nroChasis VARCHAR(50) UNIQUE,
25       seguro_id BIGINT UNIQUE NOT NULL,
26       FOREIGN KEY (seguro_id) REFERENCES SeguroVehicular(id) ON DELETE CASCADE
27   );
```

Fragmento de *01_esquema.sql*

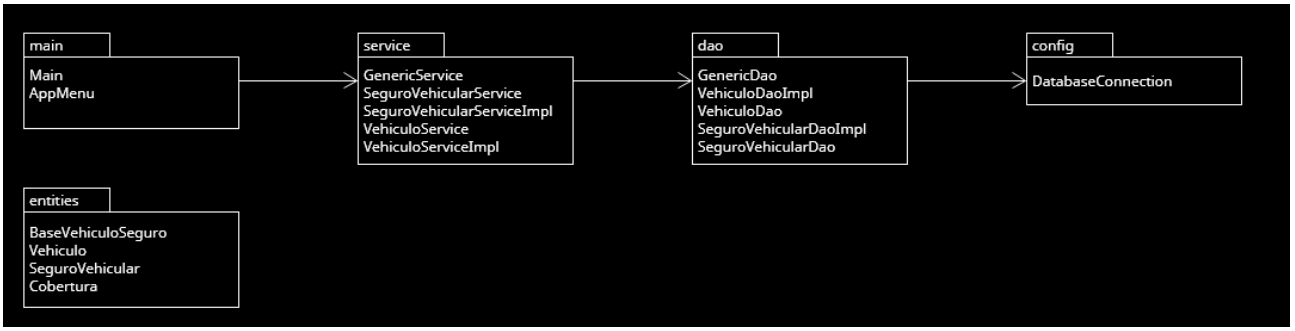
Mediante el campo booleano *eliminado*, presente en ambas entidades, se aplica el concepto de soft delete o baja lógica, con el cual se evita la eliminación física de registros, simplemente ocultándolos.

Diagramas UML

Clases:



Paquetes:



Arquitectura por capas

La aplicación está organizada en paquetes según responsabilidades:

- 📂 config/: gestión de la conexión a la base de datos.
- 📂 dao/: implementación de la persistencia con JDBC.
- 📂 entities/: contiene las clases de dominio *Vehiculo* y *SeguroVehicular*, y el enum *Cobertura*.
- 📂 service/: reglas de negocio, validaciones y manejo de transacciones.
- 📂 main/: El punto de entrada a la aplicación. Contiene Main y AppMenu

Persistencia

La persistencia del sistema se implementó mediante la **capa DAO**, que separa el acceso a datos de la lógica de negocio.. Se utilizó **MySQL 8.0** como motor de BD y **JDBC** como mecanismo de acceso desde Java.

La **Base de Datos** utiliza dos tablas: SeguroVehicular y Vehiculo:

- La relación entre ambas tablas se implementa mediante la FK *seguro_id* en la tabla Vehiculo, declarada como UNIQUE, NOT NULL y ON DELETE CASCADE. Esto garantiza que si un seguro se elimina, el vehículo asociado también debe eliminarse.
- Las dos tablas incluyen el campo booleano *eliminado*, requerido por la consigna para implementar la baja lógica, evitando así la pérdida física de datos.

En la **capa DAO** se utiliza PreparedStatement para precompilar las consultas SQL.
Las consultas SELECT filtran por eliminado = FALSE, y los DELETE se implementan como actualizaciones que marcan el registro como eliminado.
En el caso de Vehículo, se realiza un JOIN para reconstruir la composición con el seguro asociado.

Orden de operaciones y transacciones: **Capa Service** -> VehiculoServiceImpl
Para la creación de un vehículo nuevo:

- Se crea el seguro para obtener su ID.
- Ya con el ID del seguro, se puede avanzar con el vehículo.

De una manera similar, para la actualización de un vehículo:

- Se actualiza primero el seguro.
- Luego, se puede avanzar con el vehículo.

La eliminación de un vehículo se realiza con baja lógica (se oculta, no elimina) y debido a la composición, si un seguro se elimina, el vehículo asociado también desaparece. Se elimina primero el seguro, y luego se continúa con el vehiculo.

Validaciones y reglas de negocio

A través de la capa Service, se aplican las validaciones para solo persistan datos válidos y se respeten las reglas de negocios.

Vehículo

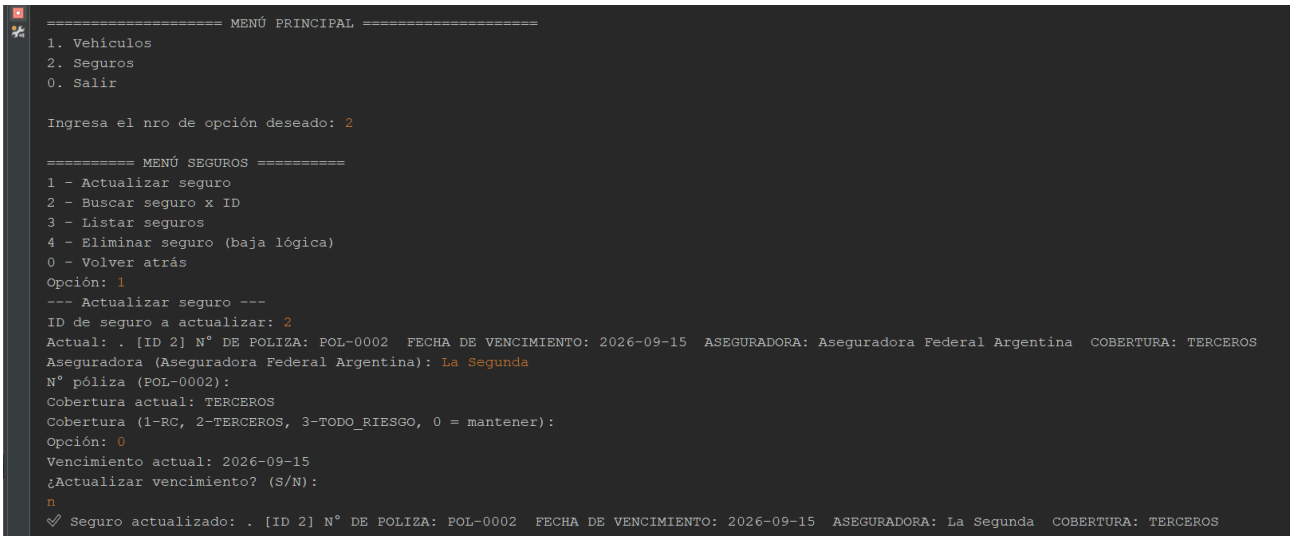
- El vehículo debe tener un seguro asociado.
- La patente es obligatoria (10 caracteres max). Marca y modelo obligatorios.
- Año válido (entre 1900 y el año actual + 1).
- Nro de chasis opcional. Si se carga, no debe superar los 50 caracteres.

SeguroVehicular

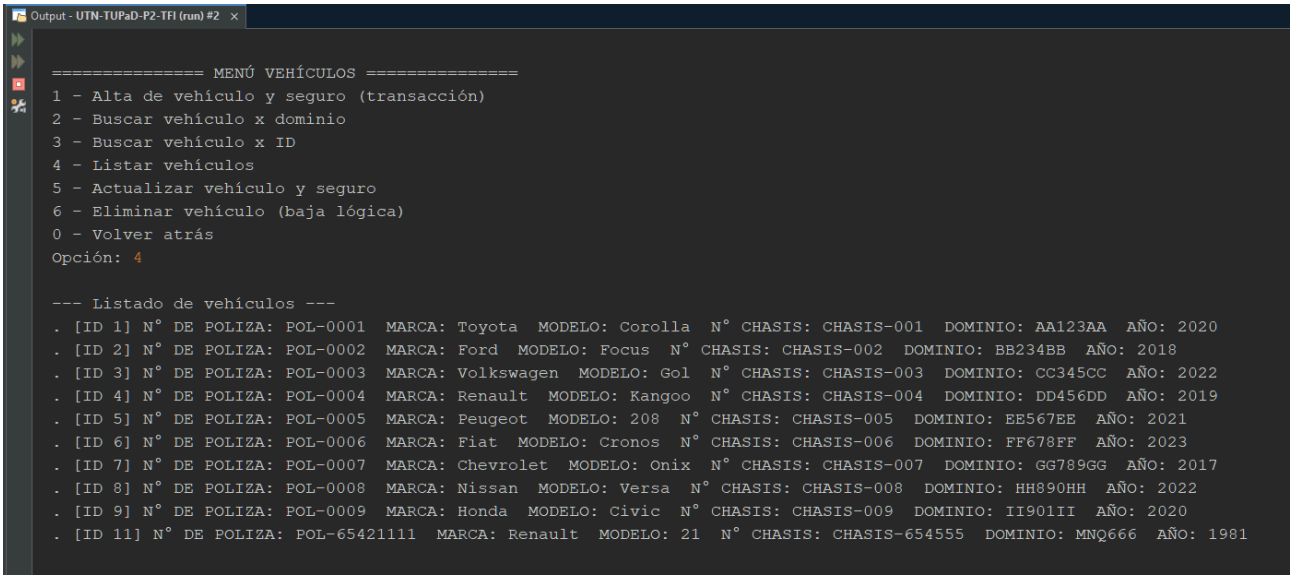
- La Aseguradora es obligatoria (80 caracteres max).
- Nro de póliza obligatorio, máximo 50 caracteres. Cobertura obligatoria.
- Fecha de vencimiento obligatoria y posterior a la actual.

Pruebas realizadas

- **Sección Seguros -> Actualizar seguro:**



- **Sección Vehículos -> Listar vehiculos:**



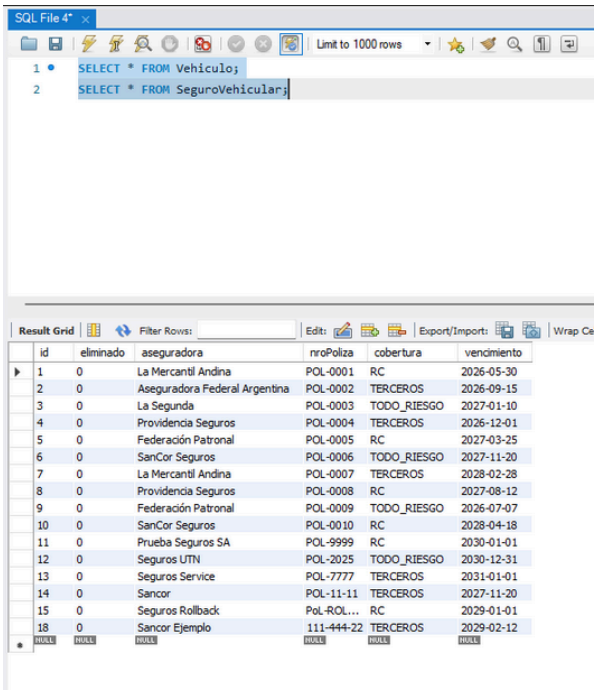
- **Simulación de rollback**

Modificamos el código para generar un error entre los INSERT que crean el seguro y el vehículo. Con eso, si pasa un error durante la carga, la excepción que incluímos para simular esto agarra el error y deshace todo:

```java
                validarVehiculo(vehiculo);
                SeguroVehicular seguro = vehiculo.getSeguro();
                if (seguro == null) {
                    throw new IllegalArgumentException("El vehículo debe tener un seguro asociado.");
                }
                validarSeguro(seguro);

                // 3) Persistir primero el seguro
                seguroDao.crear(seguro, conn);


                // ===== CÓDIGO SOLO PARA DEMO DE ROLLBACK EN EL VIDEO =====
if (true) {
    throw new RuntimeException("Falla forzada para demostrar rollback");
}
// ========================================================




                // 4) Persistir el vehiculo usando el id del seguro recien generado
                vehiculo.setSeguro(seguro);
                vehiculoDao.crear(vehiculo, conn);

                // 5) Todo ok -> commit
                conn.commit();
                return vehiculo;

            } catch (Exception e) {
                // Algo falló -> rollback
                if (conn != null) {
                    try {
                        conn.rollback();
                    } catch (SQLException ex) {
                        ex.printStackTrace();
                    }
                }
                throw new RuntimeException("Error al crear vehículo y seguro: " + e.getMessage(), e);

```

```
=============== MENU VEHICULOS ===============
1 - Alta de vehículo y seguro (transacción)
2 - Buscar vehículo x dominio
3 - Buscar vehículo x ID
4 - Listar vehículos
5 - Actualizar vehículo y seguro
6 - Eliminar vehículo (baja lógica)
0 - Volver atrás
Opción: 1
--- Alta de vehículo y seguro ---
Dominio: asflkjsa
Marca: aslkfjsa
Modelo: klasjfas
Año: 2021
Número de chasis: lkafjas
Aseguradora: akfsja
Número de póliza: 120'9321j
Cobertura (1-RC, 2-TERCEROS, 3-TODO_RIESGO):
Opción: 1
Fecha de vencimiento:
Año (YYYY): 2029
Mes (1-12): 1
1
Día (1-31): ⚠ Error: Error al crear vehículo y seguro: Falla forzada para demostrar rollback
```

Verificamos que no se agregó el registro en la base:



id	eliminado	aseguradora	nroPoliza	cobertura	vencimiento
1	0	La Mercantil Andina	POL-0001	RC	2026-05-30
2	0	Aseguradora Federal Argentina	POL-0002	TERCEROS	2026-09-15
3	0	La Segunda	POL-0003	TODO_RIESGO	2027-01-10
4	0	Providencia Seguros	POL-0004	TERCEROS	2026-12-01
5	0	Federación Patronal	POL-0005	RC	2027-03-25
6	0	SanCor Seguros	POL-0006	TODO_RIESGO	2027-11-20
7	0	La Mercantil Andina	POL-0007	TERCEROS	2028-02-28
8	0	Providencia Seguros	POL-0008	RC	2027-08-12
9	0	Federación Patronal	POL-0009	TODO_RIESGO	2026-07-07
10	0	SanCor Seguros	POL-0010	RC	2028-04-18
11	0	Prueba Seguros SA	POL-9999	RC	2030-01-01
12	0	Seguros UTN	POL-2025	TODO_RIESGO	2030-12-31
13	0	Seguros Service	POL-7777	TERCEROS	2031-01-01
14	0	Sancor	POL-11-11	TERCEROS	2027-11-20
15	0	Seguros Rollback	PoL-ROL...	RC	2029-01-01
18	0	Sancor Ejemplo	111-444-22	TERCEROS	2029-02-12
NULL	NULL	NULL	NULL	NULL	NULL

Conclusiones y mejoras futuras

Entendemos que en el desarrollo del proyecto se logró implementar una arquitectura clara en capas. Aplicamos diseño en capas, con un modelado de datos coherente e implementación sólida de CRUD con JDBC.
El resultado es una aplicación organizada, estable y fácilmente extensible.
Entre las mejoras que consideramos se podría implementar, mencionamos:

- Imprimir listados en forma de tabla.
- Más validaciones (dominio, formatos, fechas)
- Mejorar visualmente los menús (o implementar una interfaz gráfica)
- Incluir más consultas por campos relevantes (aseguradora, año, filtros por fecha de vencimiento)

Fuentes y herramientas utilizadas

- Recursos (documentación y videos) disponibles en la cursada.
- Uso de DAO en Java: Simplificando el Acceso a Datos con el patrón DAO - https://academiasanroque.com/uso-de-dao-en-java-simplificando-el-acceso-a-datos-con-el-patron-dao/
- Java Exceptions - https://www.w3schools.com/java/java_try_catch.asp
- Interface PreparedStatement - https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html
- UMLetino
- MySQL Workbench
- DBeaver CE
- Apache Netbeans
- GitHub
- ChatGPT
- Gamma AI