SEMINARIO DE PRÁCTICA DE INFORMATICA

TRABAJO PRÁCTICO 2

Alumno: Noya Hernán Ricardo

Legajo: VINF011569

Profesor Titular: PABLO ALEJANDRO VIRGOLINI

Titular experto: HUGO FERNANDO FRIAS

Contenido

Repositorio del Proyecto:

El código fuente, script SQL, diagramas y documentación complementaria del prototipo se encuentran disponibles en el siguiente repositorio de GitHub:

- **Repositorio oficial:** https://github.com/Hnoya01/SIAA-TP2-Seminario-Noya

El repositorio incluye el script SQL (NOYA-HERNAN-AP2.sql), la estructura Java en capas (MVC) y el archivo README.md con la descripción del proyecto y las instrucciones de conexión a MySQL.

Análisis

En esta etapa cierro el modelo de negocio que había presentado en el TP1 para el Sistema Integral de Autorizaciones Ambulatorias (SIAA). Decidí mantener la misma estructura de actores, reglas de negocio y flujo de estados, ya que esto me garantiza coherencia con lo trabajado anteriormente y asegura la trazabilidad hacia las próximas fases de diseño e implementación.

El alcance operativo de esta iteración se centra en una delegación y el prestador principal de la zona. Los canales disponibles son la ventanilla y el correo (registrando siempre el medio).

El ciclo de estados continúa siendo: En evaluación → Solicitar documentación → En corrección → Elevada → Autorizada / Rechazada / Anulada → Informar al afiliado → Archivada. Este flujo refuerza la trazabilidad, ya que cada transición queda registrada en la bitácora y se garantiza que antes de archivar se informe al afiliado, con la correspondiente evidencia de comunicación.

Los actores principales son:

• **Administrativo:** realiza el alta, la preevaluación, la consulta de antecedentes, la toma y cesión de casos, la comunicación y el archivo.

• **Médico Autorizador:** decide autorizar o rechazar, tiene la facultad de anular, puede solicitar documentación adicional y elevar el caso a la subgerencia.

• **Subgerencia de Prestaciones:** interviene en los casos de alto monto o complejidad, emitiendo un dictamen vinculante.

Las reglas de negocio consolidadas se sostienen tal cual las definí en el TP1. Entre ellas: vigencia máxima de órdenes de 30 días, documentación mínima obligatoria (orden + DNI), topes anuales de prácticas (por ejemplo, 30 sesiones de psicología, 25 de kinesiología), control de duplicidades en distintas ventanas temporales, y la obligación de elevar todo expediente que supere el umbral de AR$ 1.000.000. Además, las prácticas con prestadores fuera de cartilla generan una advertencia al afiliado sobre el reintegro.

En cuanto a los casos de uso, prioricé aquellos que resultan críticos para el prototipo: CU-02 Alta, CU-03 Preevaluar/Consultar antecedentes, CU-04 Decidir, CU-05 Comunicar y Archivar, y CU-01 Toma/Cesión. Esta selección la realicé porque concentran la lógica esencial del sistema y permiten mostrar un recorrido completo de una autorización desde la carga inicial hasta el archivo final.

Finalmente, incorporé el diagrama de secuencia de análisis correspondiente al caso de uso "*Decidir autorización"*. Este diagrama refleja cómo interactúan el Médico Autorizador, la interfaz del expediente, el servicio de reglas, los repositorios de solicitudes y antecedentes, y la Subgerencia de Prestaciones en caso de alto monto. Elegí este caso porque reúne los aspectos centrales: validación de reglas, control de duplicidades, elevación por monto y registro en la bitácora.

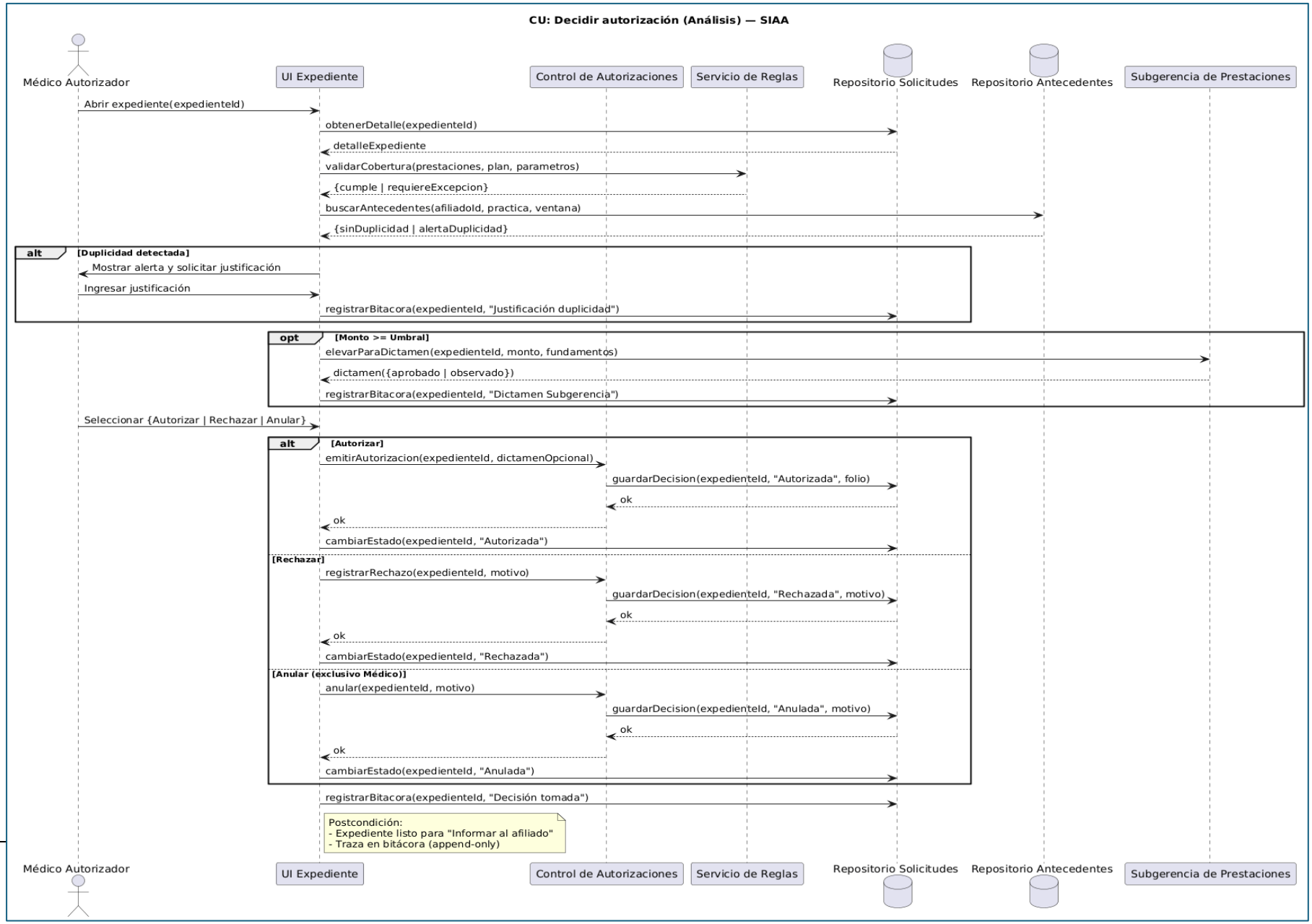

CU: Decidir autorización (Análisis) — SIAA
Médico Autorizador
UI Expediente
Control de Autorizaciones
Servicio de Reglas
Repositorio Solicitudes
Repositorio Antecedentes
Subgerencia de Prestaciones
Abrir expediente(expedienteId)
obtenerDetalle(expedienteId)
detalleExpediente
validarCobertura(prestaciones, plan, parametros)
{cumple | requiereExcepcion}
buscarAntecedentes(afiliadoId, practica, ventana)
{sinDuplicidad | alertaDuplicidad}
alt
[Duplicidad detectada]
Mostrar alerta y solicitar justificación
Ingresar justificación
registrarBitacora(expedienteId, "Justificación duplicidad")
opt
[Monto >= Umbral]
elevarParaDictamen(expedienteId, monto, fundamentos)
dictamen({aprobado | observado})
registrarBitacora(expedienteId, "Dictamen Subgerencia")
Seleccionar {Autorizar | Rechazar | Anular}
alt
[Autorizar]
emitirAutorizacion(expedienteId, dictamenOpcional)
guardarDecision(expedienteId, "Autorizada", folio)
ok
ok
cambiarEstado(expedienteId, "Autorizada")
[Rechazar]
registrarRechazo(expedienteId, motivo)
guardarDecision(expedienteId, "Rechazada", motivo)
ok
ok
cambiarEstado(expedienteId, "Rechazada")
[Anular (exclusivo Médico)]
anular(expedienteId, motivo)
guardarDecision(expedienteId, "Anulada", motivo)
ok
ok
cambiarEstado(expedienteId, "Anulada")
registrarBitacora(expedienteId, "Decisión tomada")
Postcondición:
- Expediente listo para "Informar al afiliado"
- Traza en bitácora (append-only)

Tabla de estados y disparadores

Para complementar el flujo de estados descripto en el punto anterior, presento a continuación una **tabla de estados y disparadores**. Esta herramienta nos permite visualizar de manera más clara cómo evoluciona un expediente dentro del SIAA, qué eventos generan los cambios de estado y quiénes son los actores responsables de cada transición.

Considero útil incluirla en la etapa de análisis porque refuerza el entendimiento del negocio y ayuda a garantizar que, en diseño e implementación, se respete la lógica de trazabilidad definida.

Estado actual	**Disparador / evento**	**Nuevo estado**	**Actor responsable**
En evaluación	Falta documentación	Solicitar documentación	Administrativo / Médico
Solicitar documentación	Afiliado entrega la documentación	En corrección	Administrativo
En corrección	Se cargan correcciones/adjuntos	En evaluación	Administrativo
En evaluación	Cumple reglas y antecedentes, monto < umbral → decisión del Médico	Autorizada / Rechazada	Médico Autorizador
En evaluación	Monto ≥ umbral	Elevada	Médico Autorizador
Elevada	Dictamen de Subgerencia (vinculante)	Autorizada / Rechazada	Subgerencia
Autorizada / Rechazada / Anulada	Comunicación al afiliado	Informar afiliado	Administrativo
Informar afiliado	Comunicación realizada	Archivada	Administrativo

Todas las transiciones quedan registradas en la bitácora del expediente, asegurando trazabilidad y evidencia.

De esta forma, dejamos plasmada la **dinámica del expediente** y los puntos de control más relevantes. La tabla también sirve como base para construir pruebas de validación en etapas posteriores.

Diseño

En esta etapa paso del análisis a un diseño de software concreto para el SIAA. La idea es traducir los casos de uso priorizados y las reglas del negocio en clases, servicios y repositorios, cuidando la trazabilidad con lo definido en el TP1. Decidí mantener un enfoque simple y en capas (MVC) porque me permite avanzar con el prototipo sin complejidades.

2.1 Decisiones de arquitectura (MVC en capas)

Opté por MVC con separación clara de responsabilidades:

- **UI (Presentación):** pantallas para cargar y revisar expedientes (JavaFX/Swing).
- **Aplicación / Servicios:** orquestan procesos y aplican reglas (validaciones, control de duplicidad, elevación por umbral).
- **Dominio:** objetos del negocio con su estado y comportamiento (*Solicitud*, *ItemSolicitud, Prestacion, Afiliado, Bitacora, Dictamen,* etc.).
- **Persistencia:** repositorios (interfaces) y su implementación para MySQL vía JDBC.
- **Infraestructura:** configuración de conexión, manejo de transacciones y utilitarios.

Elegí este esquema porque aísla la lógica del detalle de almacenamiento y de la UI. En pruebas puedo doblar los repositorios con implementaciones en memoria, y después conectar la misma lógica a MySQL sin tocar el resto del código.

2.2 Diagrama de clases (visión general)

El corazón del modelo es ***Solicitud***, que agrupa afiliado, prestador, estado actual, los ítems (prácticas) y la bitácora (traza). Para mantener flexibilidad, cada práctica va en un ***ItemSolicitud*** con su cantidad y **precio unitario "snapshot"** (guardado al momento del alta), y el total se calcula a partir de esos ítems. De esta manera, si el precio de una prestación cambia a futuro, no se alteran históricos.

Entidades principales

- ***Solicitud***: *id, fecha, estado, (total calculado)*; relación con ***Afiliado***, ***Prestador***, ***ItemSolicitud***, ***Bitacora*** y ***Dictamen***.
- ***ItemSolicitud****: cantidad,* ***precio_Unit*** (snapshot), referencia a ***Prestacion****; subtotal() = cantidad * precio_Unit*
- ***Prestacion****: codigo, descripcion, precio (precio "vigente"; se usa como valor por defecto al crear el ítem)*.
- ***Afiliado****: dni, nombre, plan.*
- ***Prestador****: id, nombre.*
- ***Bitacora*** *(append-only)*: *fecha,* ***actorId***, ***rol***, *evento, detalle.*
- ***Dictamen****: resultado, observaciones,* ***fecha***, ***emitidoPor*** *(usuario de Subgerencia)*.
- ***EstadoSolicitud*** *(enum)*: *EN_EVALUACION, SOLICITAR_DOC, EN_CORRECCION, ELEVADA, AUTORIZADA, RECHAZADA, ANULADA, INFORMAR_AFILIADO, ARCHIVADA*.

Servicios y repositorios

- ***ControlAutorizaciones****:* coordina la decisión (autorizar / rechazar / anular) y **asegura la traza**.
- ***ServicioReglas****:* valida cobertura, topes/ventanas y si corresponde **elevación por umbral**.
- ***ServicioAntecedentes****:* consulta **duplicidad** por ventanas de tiempo.
- **Repositorios (interfaces desacopladas de MySQL)**: ***RepositorioSolicitudes, RepositorioAntecedentes, RepositorioParametros****.*

Nota operativa: en decisiones (autorizar/rechazar/anular) **registro bitácora y cambio de estado en una única transacción**. Esto garantiza consistencia y simplifica las pruebas.

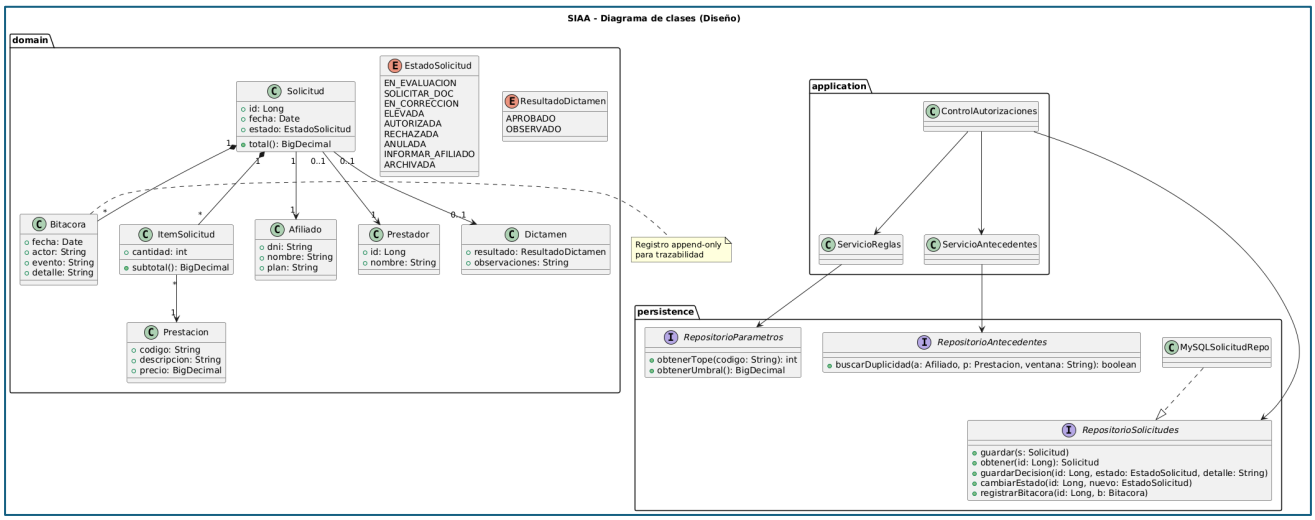


Diagrama de clases

2.3 Responsabilidades y contratos

Para que el diseño sea operable en el prototipo, definí **contratos sintéticos**. No son la API final, pero me sirven como guía al implementar y para asegurar que la lógica del sistema quede bien delimitada.

- **ControlAutorizaciones**
  - *emitirAutorizacion(solicitud, dictamenOpcional)*
  - *registrarRechazo(solicitud, motivo)*
  - *anular(solicitud, motivo)*
  - *Invariante:* cada decisión registra bitácora y cambia estado **en una única transacción**.
- **ServicioReglas**
  - *validarCobertura(solicitud) → ResultadoReglas*
  - Considera topes y ventanas de tiempo, e indica si corresponde **elevación por monto**.

- **ServicioAntecedentes**
    - *buscarDuplicidad(afiliado, prestacion, ventana) → ResultadoDuplicidad*
    - Retorna no solo el booleano, sino también métricas básicas (cantidad de antecedentes, última fecha), lo que me permitirá enriquecer la interfaz.
- **RepositorioSolicitudes (interface)**
    - *guardar(solicitud)*
    - *obtener(id)*
    - *registrarDecision(id, nuevoEstado, detalle) (unifica cambio de estado + bitácora para decisiones atómicas)*
    - *registrarBitacora(id, bitacora) (para eventos no decisorios)*
- **RepositorioAntecedentes (interface)**
    - *buscarDuplicidad(afiliado, prestacion, ventana)*
- **RepositorioParametros (interface)**
    - *obtenerTope(codigo)*
    - *obtenerUmbral()*

Definí los repositorios como **interfaces** para poder intercambiar MySQL por dobles de prueba sin reescribir los servicios. Esto me da flexibilidad en pruebas y asegura que la lógica de negocio no dependa de la tecnología de persistencia.

2.4 Diagrama de despliegue (conexión a MySQL)

Mostré dos escenarios compatibles con el prototipo:

1. **Local**: la app Java y MySQL corren en la misma PC. Es el modo más práctico para las pruebas iniciales, ya que simplifica la configuración.
2. **Cliente–Servidor**: la app Java corre en los puestos (Administrativo y Médico) y MySQL en un servidor de red (puerto TCP 3306). La aplicación lee host, puerto, esquema y credenciales desde un archivo de configuración (config.properties) y se conecta vía JDBC con un **usuario de privilegios mínimos**.

En este escenario, también configuré reglas operativas básicas:

- El firewall abre el puerto 3306 únicamente para la subred de los puestos.
- La conexión JDBC se configura con **SSL/TLS** para proteger datos en tránsito.
- Los backups de la base de datos quedan programados en el servidor.

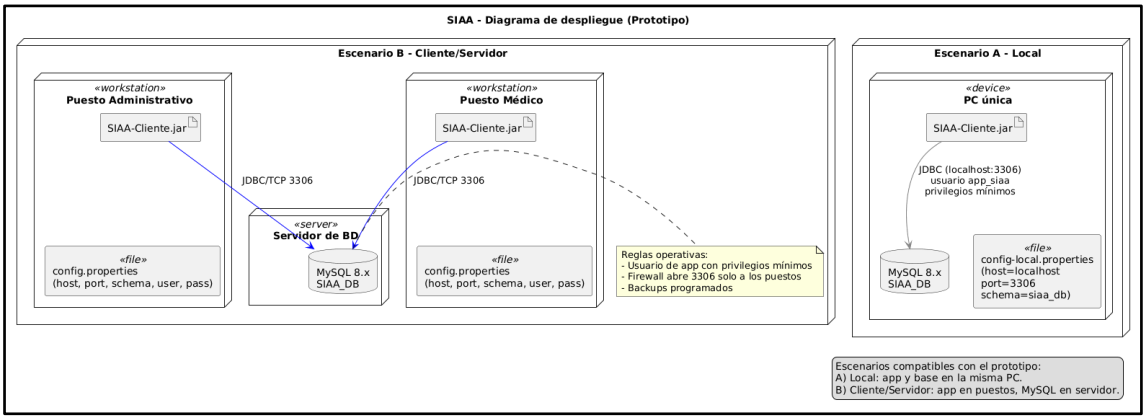


Diagrama de despliegue

2.5 Organización del código (paquetización sugerida)

Para mantener orden en la implementación, **definí la siguiente estructura de paquetes**:

- ***ui.**** → pantallas y controladores de la interfaz de usuario.
- ***application.**** → servicios y orquestación de procesos.
- ***domain.**** → entidades, enums y lógica de dominio.
- ***persistence.**** → interfaces de repositorios e implementaciones JDBC para MySQL.
- ***infrastructure.**** → configuración de conexión, manejo de transacciones y utilitarios.

Elegí esta organización porque es simple, legible y evita dependencias cruzadas innecesarias. Cada capa conoce solo a la inmediatamente inferior, lo que facilita pruebas y posibles cambios de tecnología sin comprometer el diseño.

Cierre de la sección

Con este diseño dejé definidos los objetos del dominio, las reglas que los gobiernan y los puntos de integración con MySQL. La elección de MVC en capas me permite avanzar con un prototipo operativo en Java, sosteniendo la trazabilidad con el análisis y preparando el terreno para la siguiente etapa: 3) Implementación.

3. Implementación

En esta etapa presento cómo el diseño se trasladará a un prototipo operable en Java con persistencia en MySQL. El objetivo no es mostrar código final, sino explicar cómo se implementará el modelo y qué decisiones permitirán que los casos de uso priorizados puedan ejecutarse de forma completa. Mantengo la alineación con el Proceso Unificado de Desarrollo (PUD) indicado por la cátedra, asegurando trazabilidad entre análisis, diseño, implementación y pruebas.

3.1 Arquitectura a utilizar

Adopté una arquitectura en capas tipo MVC por su claridad y organización:

- Interfaz (JavaFX o Swing): muestra y recibe datos.
- Servicios: coordinan procesos y aplican reglas (validaciones, duplicidad, decisiones).
- Dominio: contiene las entidades principales *(Solicitud, Afiliado, Prestacion, Bitacora, Dictamen, etc.).*
- Persistencia: implementa repositorios para la conexión con MySQL.
- Infraestructura: gestiona la configuración y la conexión.

Definí los repositorios como interfaces, lo que me permite probar la lógica en memoria y luego conectarla a MySQL sin modificar el resto del sistema.

3.2 Del modelo conceptual al lógico

Las clases del dominio se traducirán a tablas en MySQL:

- *Solicitud* → tabla *solicitud*.
- *ItemSolicitud* → tabla *item_solicitud* (con **precio_unitario** para mantener snapshot histórico).
- *Bitacora* → tabla propia, en modo **append-only** para trazabilidad.
- *Dictamen* → tabla vinculada a *solicitud*.
- *EstadoSolicitud* y *ResultadoDictamen* → columnas con restricción CHECK / ENUM controlado.

3.3 Implementación de los casos de uso

Los casos priorizados se cubrirán de la siguiente forma:

- **Alta**: el administrativo registrará la solicitud inicial.
- **Preevaluar / Antecedentes**: se aplicarán reglas y se consultará duplicidad.
- **Decidir**: el médico autorizará, rechazará o anulará; si el monto supera el umbral, se solicitará dictamen a la Subgerencia.
- **Comunicar y archivar**: se registrará la notificación y se marcará como archivada.

- **Toma y cesión**: se gestionará la concurrencia entre usuarios.

3.4 Manejo de transacciones y errores

Cada decisión se ejecuta dentro de una **transacción** que incluye validación, registro en bitácora, cambio de estado y confirmación. En caso de error, se aplica **rollback**. La concurrencia se maneja con toma/cesión y control optimista para evitar pisadas.

3.5 Conexión a la base

La conexión con MySQL se resuelve mediante un archivo externo (*config.properties*) con host, puerto, esquema y credenciales. Uso *PreparedStatement* para prevenir inyecciones y un usuario con privilegios mínimos para la aplicación. En el escenario cliente-servidor, configuro JDBC con SSL/TLS para asegurar datos en tránsito. Los enums de estado se convierten entre Java y SQL.

3.6 Decisiones previstas

- Utilizo **JDBC directo** en lugar de un ORM, para mantener simplicidad en el prototipo.
- La bitácora será **append-only**.
- Los enums se mantienen en Java y en SQL para evitar valores inválidos.
- Los repositorios como interfaces permiten intercambiar implementaciones sin afectar la lógica.

Cierre de la sección

Con estas decisiones obtendré un prototipo sencillo pero funcional, capaz de cubrir los casos de uso críticos, mantener la trazabilidad y gestionar la conexión a la base de forma desacoplada y configurable. El script SQL completo lo entregaré como archivo .sql/.txt independiente; en el informe incluiré 5–6 capturas representativas (creación, inserción, consulta) para evidenciar la ejecución real.

4. Pruebas

En esta etapa defino cómo voy a poner a prueba el prototipo para asegurar que cumpla las reglas de negocio, los casos de uso priorizados y los criterios de aceptación (CA) del TP1. El foco está en el flujo crítico de autorizaciones y en los indicadores del MVP; no intento cubrir integraciones o portales que están fuera de alcance.

4.1 Alcance y enfoque

Voy a probar los CU priorizados: **Alta**, **Preevaluar/Antecedentes**, **Decidir**, **Comunicar/Archivar** y **Toma/Cesión**.

El enfoque es escalonado:

1. **Unitarias (JUnit)** sobre dominio y servicios: cálculo de totales, cambios de estado válidos y validación de reglas/topes/ventanas.
2. **Integración** con repositorios (dobles en memoria y MySQL real).
3. **End-to-end** recorriendo los CU en la UI (JavaFX/Swing + JDBC).
4. **Aceptación** contra los CA del TP1 y **medición del SLA** del MVP.

Además, incluyo verificaciones **no funcionales básicas**: control de **seguridad por roles** (quién puede decidir o anular) y **rendimiento mínimo** para cumplir los tiempos del SLA.

4.2 Trazabilidad

Para mantener la trazabilidad armé una **mini-matriz** que vincula cada caso de uso (CU) con sus criterios de aceptación (CA del TP1), las reglas o KPI que impacta y lo que se valida en las pruebas:

CU	**CA (TP1)**	**Regla/KPI**	**Qué se valida**
Alta	CA-01, CA-07	Doc. mínima, adjuntos válidos	Alta con Orden + DNI; rechazo de adjuntos inválidos
Preevaluar/Antecedentes	CA-03, CA-08	Duplicidad, faltantes	Bloqueo por duplicidad y paso a “Solicitar documentación”
Decidir	CA-02, CA-05, CA-09	Topes, umbral, anulación	Autorización simple; elevación obligatoria; anulación solo Médico
Comunicar/Archivar	CA-06	Comunicación obligatoria	No archiva sin evidencia de notificación
Toma/Cesión	CA-04	Concurrencia	Aviso de “tomado por otro” y cesión controlada
Métrica	—	SLA 48h (p95 ≤ 72h)	Medición sobre expedientes del MVP

4.3 Datos y entorno de prueba

Para las pruebas voy a usar un **entorno específico** con la aplicación en Java (UI) y MySQL 8.x, sobre un esquema exclusivo llamado siaa_test.

Preparé un **dataset sintético** con distintos escenarios:

- Afiliados con autorizaciones previas **dentro y fuera de la ventana** (29/30/31 días) para probar bordes en la regla de duplicidad.
- Prestaciones de **alto monto** justo en los valores de borde (**igual, −1 y +1** respecto del umbral).
- Adjuntos **válidos** (PDF/JPG/PNG) y adjuntos **rechazados** (mayores a 5 MB o formatos no permitidos).
- Al menos **un prestador fuera de cartilla** para validar el mensaje informativo al afiliado.

Además, verifico que **toda acción quede registrada en la bitácora** (append-only), con fecha, actor y rol.

Finalmente, aplico **buenas prácticas en MySQL**: sql_mode estricto y time_zone fija, para que las validaciones de ventanas temporales no se vean afectadas por configuraciones regionales.

4.4 Casos de prueba representativos (Given–When–Then)

A continuación detallo los casos de prueba más relevantes. Todos se presentan en formato **Given–When–Then** para dejar clara la condición inicial, la acción y el resultado esperado.

- **CP-01 Alta mínima (CA-01)**
  Dado un administrativo con DNI, plan, orden y al menos una prestación válida,
  cuando confirma el alta,
  entonces la solicitud queda en estado **En evaluación** y se registra en la bitácora el evento “Alta”.
- **CP-02 Adjuntos válidos/Inválidos (CA-07)**
  Dado que se adjuntan archivos,
  cuando el formato o tamaño no cumplen con las condiciones,
  entonces el sistema rechaza el adjunto;
  y con archivos válidos los acepta y deja registro en la bitácora.
- **CP-03 Duplicidad sin justificación (CA-03)**
  Dado un afiliado con una autorización previa dentro de la ventana,
  cuando se intenta emitir sin justificación,
  entonces se bloquea la decisión;
  y si se agrega justificación, el sistema permite continuar y registra la acción en la bitácora.
- **CP-04 Elevación por monto (CA-05)**
  Dado una solicitud con monto superior al umbral,
  cuando el Médico intenta decidir,

entonces el sistema exige la elevación a Subgerencia;
y con dictamen **Aprobado/Observado**, la solicitud pasa a **Autorizada** o **Rechazada**.

- **CP-05 Umbral en el borde**
  Dado una solicitud con monto exactamente igual al umbral,
  cuando el Médico decide,
  entonces valido el comportamiento configurado (elevar o no), y documento el resultado como evidencia.

- **CP-06 Comunicar antes de archivar (CA-06)**
  Dado una solicitud en estado Autorizada/Rechazada/Anulada,
  cuando se intenta archivar,
  entonces el sistema exige evidencia de notificación al afiliado;
  y si no está cargada, bloquea el cierre.

- **CP-07 Concurrencia (CA-04)**
  Dado que un expediente está tomado por un usuario,
  cuando otro intenta editarlo,
  entonces el sistema muestra un aviso y solo permite acceso en lectura o la solicitud de cesión.
  (Con control optimista por campo version verifico que no haya pisadas de datos).

- **CP-08 Permisos de anulación**
  Dado un usuario que no es Médico,
  cuando intenta anular una solicitud,
  entonces el sistema deniega la acción y registra el intento en la bitácora.

Para cada caso de prueba realizo una **doble verificación**: en la **UI** (estado y mensajes mostrados) y en la **base de datos** (estado y eventos registrados en la bitácora).

4.5 Evidencia y resultados esperados

Considero que un caso de prueba está **aprobado** si cumple con los criterios de aceptación (CA) definidos en el TP1.

La evidencia se compone de:

- **Capturas de la UI**, mostrando estados y mensajes relevantes.
- **Consultas SQL** que confirman el estado de la solicitud y el registro en la bitácora, por ejemplo:

```sql
SELECT estado
  FROM solicitud
 DÓNDE identificación = :identificación;

SELECT fecha, actorId, rol, evento, detalle
  DE bitácora
 DONDE solicitud_id  = :id
 ORDENAR POR FECHA;
```

Para los bordes de duplicidad:

```sql
SELECT COUNT(*) AS cant, MAX(fecha) AS ultima
  DESDE autorizaciones_hist
 DÓNDE afiliado_dni = :DNI
   Y prestacion_codigo = :código
   Y fecha BETWEEN :desde Y :hasta;
```

Además, mido el **SLA del MVP**: al menos el 90% de las solicitudes deben resolverse en ≤48h, con un percentil 95 ≤72h, usando un set controlado de expedientes de prueba.

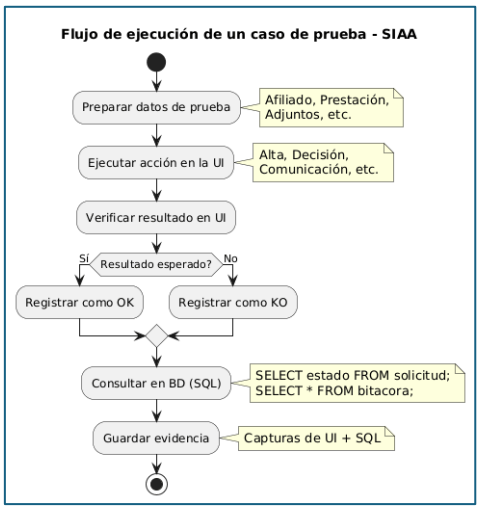
4.6 Riesgos y mitigación

Durante las pruebas identifiqué algunos riesgos que podrían afectar la validez de los resultados o la repetibilidad de los casos. Para cada uno definí una acción de mitigación:

- **Parámetros mal cargados (topes, umbral, ventanas):** pueden generar falsos rechazos o aceptaciones. Para mitigarlo, ejecuto **pruebas de bordes** (−1, =, +1 respecto de cada parámetro) antes de cada corrida.
- **Datos de cartilla o planes desactualizados:** podrían distorsionar la validación de duplicidad o cobertura. Definí la **actualización mensual del dataset de prueba** como control preventivo.
- **Interferencias entre pruebas:** la reutilización de datos puede producir resultados inconsistentes. Para evitarlo, hago un **reset completo del esquema siaa_test en cada corrida** mediante un script de drop/create/seed.
- **Accesos indebidos a antecedentes:** representan un riesgo de seguridad. Para mitigarlo, limito los **permisos por rol** al mínimo necesario y mantengo una **bitácora de accesos**.
- **Flujos dependientes de horario:** diferencias en zonas horarias pueden afectar validaciones de ventanas. Mitigo este riesgo configurando una **time_zone fija en MySQL** y utilizando **fechas controladas** en los seeds.

Diagrama de apoyo

El flujo de ejecución de un caso de prueba (UI + SQL + evidencia) se muestra en la figura siguiente, donde se integran los pasos de preparación de datos, ejecución, validación y registro de resultados.



5. Base de datos MySQL

En esta etapa defino cómo queda la base de datos del sistema. El objetivo es traducir el modelo de clases y las reglas de negocio a un esquema relacional en **MySQL 8.x**, cuidando la **normalización**, la **integridad referencial** y la **trazabilidad** de cada acción.

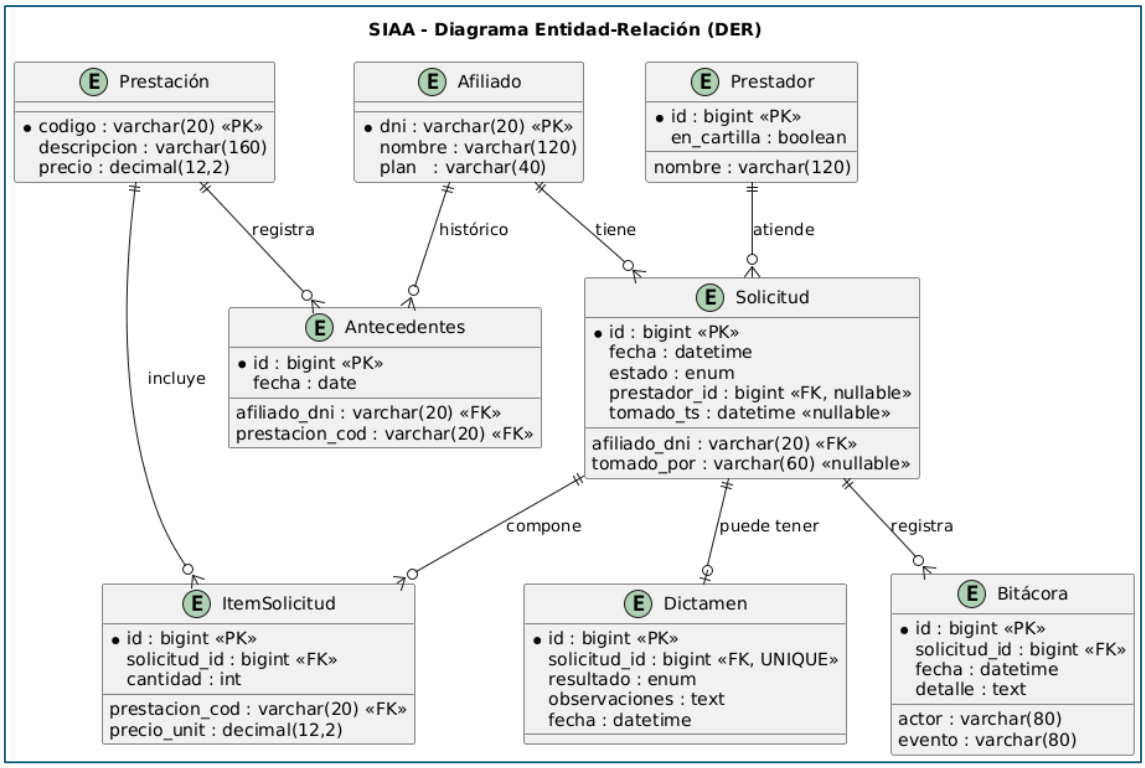
5.1 Criterios de diseño

Para organizar el esquema seguí estos criterios principales:

- **3FN:** cada entidad (afiliado, prestador, prestación, solicitud, ítems, bitácora, dictamen) se almacena sin duplicar información.
- **Bitácora append-only:** la tabla solo agrega registros; no hay UPDATE ni DELETE, lo que garantiza una auditoría completa.
- **Estados y dictámenes como ENUM:** simplifica el prototipo y evita valores inválidos, manteniendo un mapeo 1:1 con los enum de Java.
- **Dictamen 1:1 opcional con Solicitud:** solo existe cuando hay elevación a Subgerencia (un dictamen por solicitud).
- **Tabla de antecedentes:** agiliza el control de duplicidades dentro de una ventana temporal.
- **Índices útiles:** definidos sobre los campos consultados con más frecuencia (afiliado, estado, fechas y claves foráneas).
- **Buenas prácticas:** uso de utf8mb4, motor **InnoDB**, sql_mode estricto y time_zone fija, para asegurar consistencia en las reglas de validación temporal.

Como mejora opcional, evalué la incorporación de un campo version (INT) en la tabla solicitud, con el fin de aplicar control optimista y fortalecer el mecanismo de “toma/cesión” en escenarios concurrentes.

5.2 DER propuesto



Las relaciones clave son:

- Un **Afiliado** puede tener varias **Solicitudes**.
- Una **Solicitud** puede estar vinculada a un **Prestador** y contiene varios **ítems**.
- Cada **ItemSolicitud** referencia a una **Prestación** y guarda el campo *precio_unit* (snapshot) para preservar históricos.
- Cada **Solicitud** genera entradas en **Bitácora** y, opcionalmente, un **Dictamen** en relación 1:1.
- La tabla **Antecedentes** permite verificar duplicidades de prestaciones en una ventana temporal.

Además, para reforzar la integridad y el rendimiento definí:

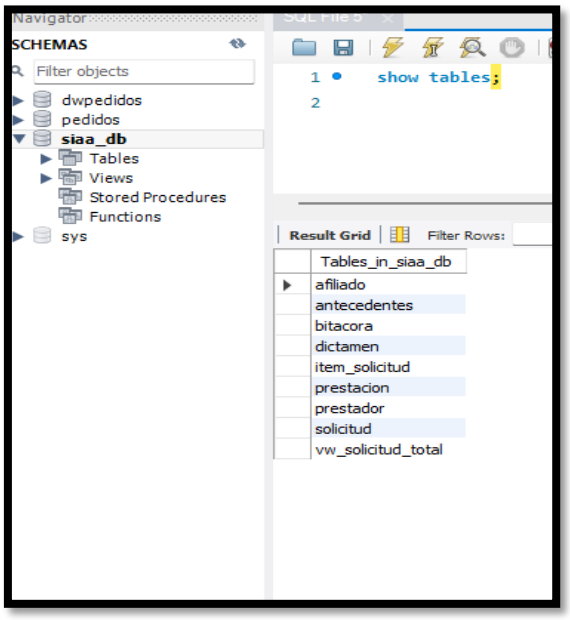
- **ON DELETE RESTRICT** entre *solicitud y bitacora/item_solicitud* (no quiero perder auditoría ni ítems por borrados accidentales).
- **Índices compuestos**:
    - *antecedentes(afiliado_dni, prestacion_cod, fecha)*
    - *bitacora(solicitud_id, fecha)*
    - *item_solicitud(solicitud_id)*
    - *solicitud(estado, fecha)*

5.3 Script SQL

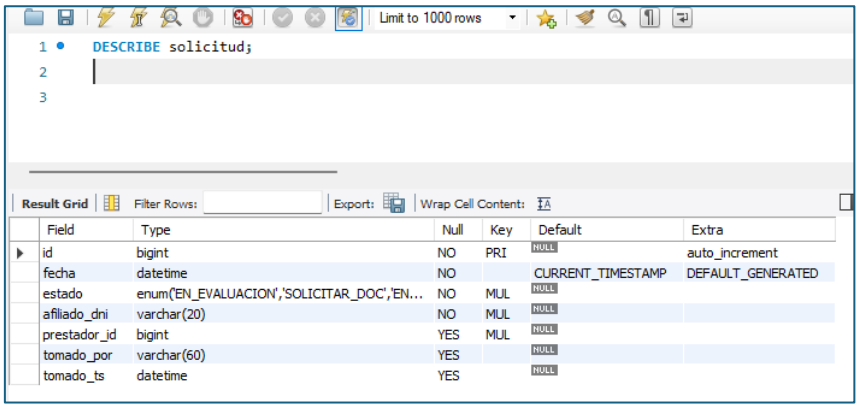
El esquema completo lo entrego como archivo aparte **NOYA-HERNA-AP2.sql**. En el informe solo incluyo algunas capturas representativas que muestran la creación de tablas, inserciones y consultas de verificación.

El script contiene:

- Creación del esquema *siaa_db*.
- Tablas: *afiliado, prestador, prestacion*.
- Tablas de proceso: *solicitud* y *item_solicitud* (con campo *precio_unit*).
- Tabla de **Bitácora** (append-only).
- Tabla de **Dictamen** (1:1 opcional con solicitud, *UNIQUE (solicitud_id)*).
- Tabla de **Antecedentes** para control de duplicidad.
- **Índices** en campos críticos.
- Una **vista** *vw_solicitud_total* para calcular los totales (*SUM(cantidad * precio_unit)*).
- Datos de prueba mínimos (afiliados, prestadores, prestaciones).
- Ejemplo de **alta de solicitud** con ítems y registro en la bitácora.



Tablas creadas en el esquema (SHOW TABLES)

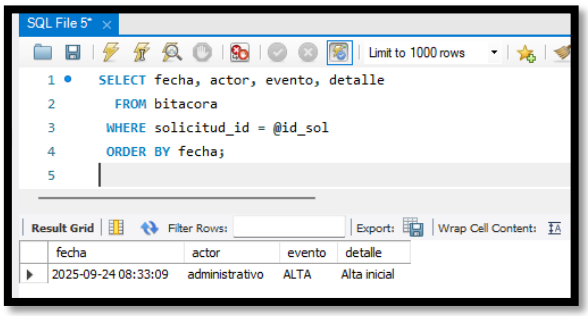


Estructura de solicitud (DESCRIBE)

5.4 Consultas clave para evidencia

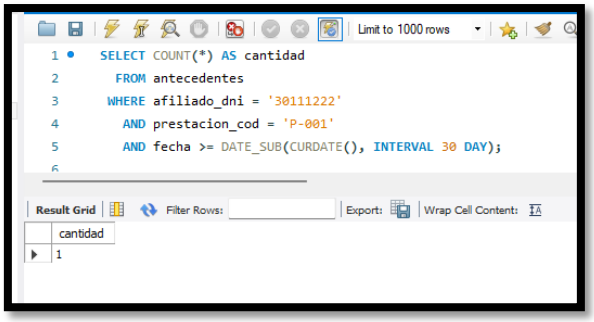
Estas son las consultas que utilizo como evidencia y que ya se muestran en las capturas:

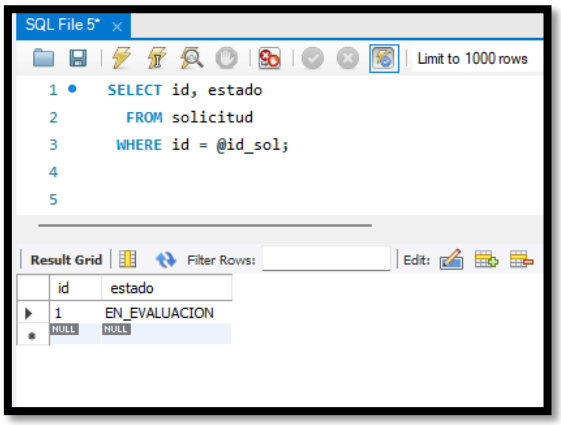
- *SHOW TABLES;* → evidencia las tablas creadas.
- *DESCRIBE solicitud*; → muestra la estructura de la entidad principal.
- *SELECT estado FROM solicitud WHERE id = :id*; → verifica cambios de estado



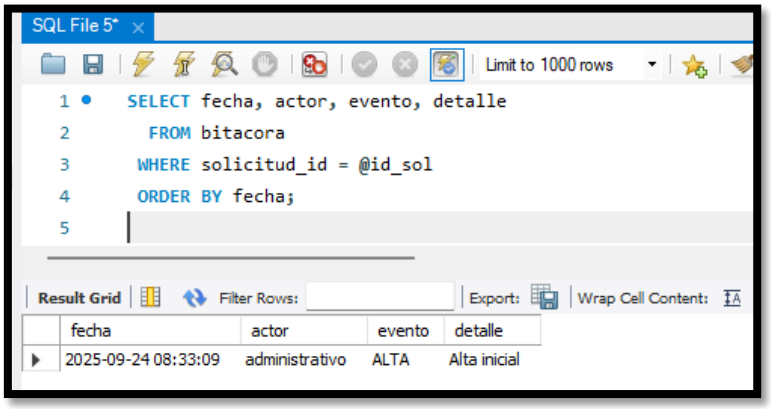
Trazabilidad en la bitácora.

- *SELECT * FROM vw_solicitud_total WHERE id = :id;* → cálculo del total por vista.
- **Duplicidad en 30 días:**

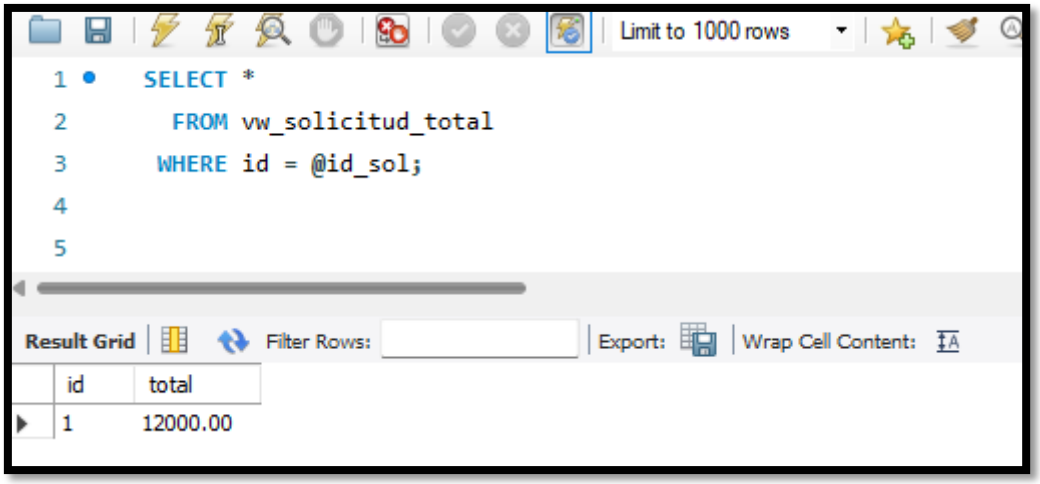




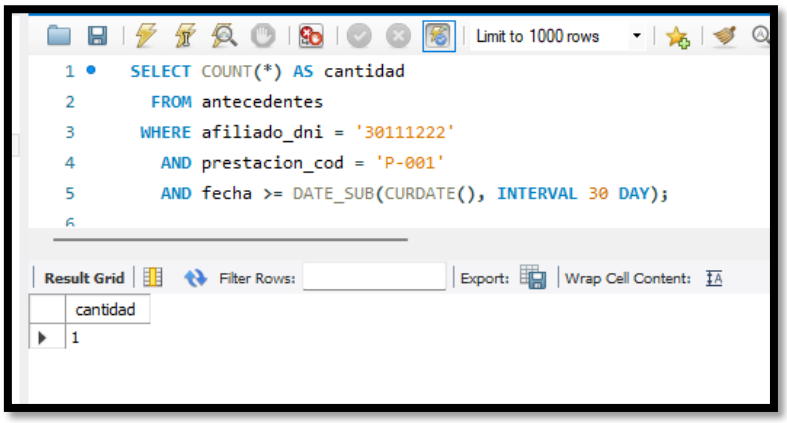
Estado actual de la solicitud (SELECT estado)



*Bitácora registrada (SELECT * FROM bitacora)*



Totales calculados por vista (vw_solicitud_total)



Número de antecedentes dentro de la ventana de 30 días

5.5 Decisiones de diseño

- **ENUM en lugar de tablas de estados:** para este prototipo no necesito parametrización dinámica, por lo que opté por *ENUM*. Esto me permite evitar joins adicionales y mantener un mapeo directo y simple con los *enum* definidos en Java.

- **Tabla de antecedentes explícita:** aunque en un sistema productivo se podría derivar de las autorizaciones ya emitidas, preferí crear una tabla dedicada. De esta forma agilizo el control de duplicidad en ventanas de tiempo y simplifico las consultas.

- **Totales calculados por vista:** resolví el cálculo de montos con una vista (*vw_solicitud_total*). Así no almacena redundancia y evito inconsistencias entre datos parciales, manteniendo siempre la información derivada al día.

Cierre de la sección

Con este diseño la base de datos del **SIAA** queda lista para soportar el prototipo, garantizando **normalización**, **integridad referencial** y **trazabilidad** a través de la bitácora. El **script SQL** se entrega como archivo aparte, mientras que en el informe incluyo solo las capturas necesarias para evidenciar su ejecución real.

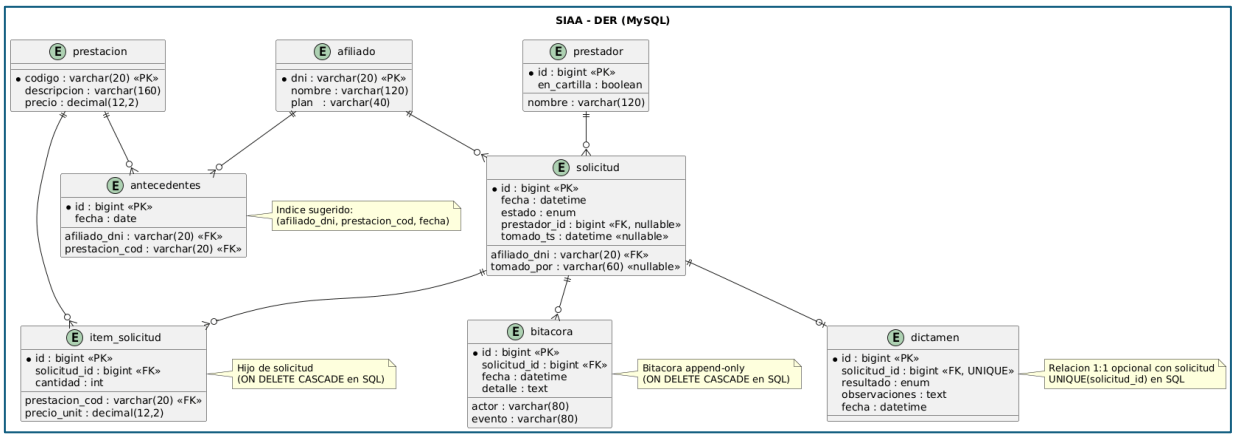
6. Diagrama DER

Objetivo

Presentar el modelo relacional del **SIAA** de forma gráfica, mostrando entidades, claves y relaciones tal como se implementarán en **MySQL**. La intención es que el DER refleje lo ya definido en las etapas de **Análisis/Diseño** y mantenga la **trazabilidad** con el script SQL entregado aparte.

Explicación

El esquema se organiza en torno a la entidad **solicitud**, que constituye el núcleo del proceso. Cada solicitud pertenece a un **afiliado**, puede estar vinculada a un **prestador** y se compone de varios **ítems** que referencian a las **prestaciones**. La **bitácora** registra cada evento en modo *append-only*, mientras que el **dictamen** modela la intervención de Subgerencia en los casos de elevación (relación **1:1 opcional**). Por su parte, la tabla **antecedentes** facilita el control de **duplicidad** de prestaciones dentro de una ventana temporal.



7. Creación de las tablas MySQL

Para materializar el modelo en la base, utilicé el script **APELLIDO-NOMBRE-AP2.sql**. El script crea el esquema **siaa_db**, define **todas las tablas con sus claves/índices**, y agrega la **vista de totales**. Trabajé con **MySQL 8.x**, motor **InnoDB**, utf8mb4, sql_mode estricto y time_zone fija para mantener consistencia.

Orden de creación (para evitar errores de FK)

1. Tablas maestras: *afiliado, prestador, prestacion.*
2. Proceso: *solicitud*.
3. Hijas: *item_solicitud, bitacora, dictamen.*
4. Soporte: *antecedentes*.
5. Vistas e índices extra.

Reglas de integridad y performance (resumen)

- **PK y AUTO_INCREMENT** en las tablas con *id* (p. ej., *solicitud, item_solicitud, bitacora, dictamen, antecedentes*).
- **FKs** definidas con:
    - *solicitud(afiliado_dni) → afiliado(dni)*
    - *solicitud(prestador_id) → prestador(id) (nullable)*
    - *item_solicitud(solicitud_id) → solicitud(id)*
    - *item_solicitud(prestacion_cod) → prestacion(codigo)*
    - *bitacora(solicitud_id) → solicitud(id)*
    - *dictamen(solicitud_id) → solicitud(id) (UNIQUE para 1:1 opcional)*
    - *antecedentes(afiliado_dni, prestacion_cod) → afiliado/prestacion*
- **Acciones de borrado** (criterio del prototipo):
    - *ON DELETE CASCADE* en ***item_solicitud*** *y* ***dictamen*** para simplificar pruebas.
    - **Nota de auditoría:** si bien podría dejar *CASCADE* en ***bitacora*** para el entorno de prueba, en un escenario productivo corresponde **RESTRICT/NO ACTION** para **no perder trazabilidad**.
- **Defaults/Checks** recomendados (y simples de implementar en MySQL 8):
    - *solicitud.estado* ***DEFAULT EN_EVALUACION***
    - *solicitud.fecha, bitacora.fecha* ***DEFAULT CURRENT_TIMESTAMP***
    - *item_solicitud.cantidad > 0, item_solicitud.precio_unit >= 0 (CHECK)*
    - *prestador.en_cartilla BOOLEAN NOT NULL DEFAULT 1*
- **Índices útiles**:

- *solicitud(afiliado_dni, estado, fecha)*
- *item_solicitud(solicitud_id)*
- *bitacora(solicitud_id, fecha)*
- *antecedentes(afiliado_dni, prestacion_cod, fecha)*

Verificación rápida en MySQL (evidencia)

Comandos que corrí para comprobar que todo quedó correcto:

```
SHOW TABLES;                                    -- listado de tablas

DESCRIBE solicitud;                             -- estructura de la tabla principal

SHOW FULL TABLES WHERE Table_type = 'VIEW'; -- verificar que la vista existe

SELECT *
  FROM vw_solicitud_total
 WHERE id = :id;                                -- chequeo de la vista de totales
```

8. Inserción, consulta y borrado de registros

En este apartado muestro operaciones básicas sobre la base (**insertar, consultar y limpiar datos de prueba**) para evidenciar que el esquema funciona como espero. Trabajo siempre con **datos de prueba** y aclaro que, en el sistema final, los expedientes **no se eliminan**: se **archivan** para preservar la trazabilidad.

Ajustes aplicados al esquema (coherencia con TP1)

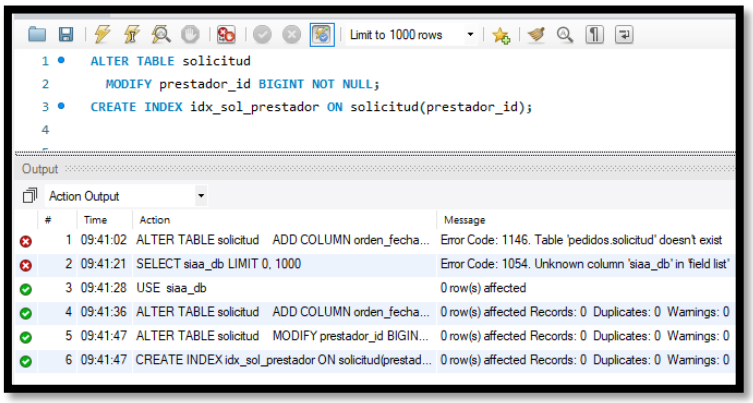
Para alinear la BD con lo definido en el TP1, incorporé dos ajustes simples:

- **Fecha de orden médica** en solicitud (control de vigencia ≤ 30 días):

```
ALTER TABLE solicitud
  ADD COLUMN orden_fecha DATE NULL AFTER fecha;
-- (Opcional, cuando ya se cargue en todos los casos)
-- ALTER TABLE solicitud MODIFY orden_fecha DATE NOT NULL;
```

- **Prestador obligatorio** en el alta + **índice** para búsquedas:

```
ALTER TABLE solicitud
  MODIFY prestador_id BIGINT NOT NULL;

CREATE INDEX idx_sol_prestador ON solicitud(prestador_id);
```



Evidencia de ALTERs e índice

A) Inserción (datos de prueba)

Cargo registros mínimos para simular un caso real: afiliado, prestador, prestaciones, alta de solicitud con ítems y su bitácora.

```sql
-- Afiliado y prestador de prueba
SET @af := '40111222';
INSERT INTO afiliado (dni, nombre, plan) VALUES (@af, 'Carla Diaz', 'P1')
ON DUPLICATE KEY UPDATE nombre=VALUES(nombre), plan=VALUES(plan);

INSERT INTO prestador (nombre, en_cartilla) VALUES ('CENTRO SALUD NORTE', TRUE);
SET @prest := LAST_INSERT_ID();

-- Prestación de alto monto para probar elevación
INSERT INTO prestacion (codigo, descripcion, precio)
VALUES ('P-999', 'Tratamiento especial', 1200000.00)
ON DUPLICATE KEY UPDATE descripcion=VALUES(descripcion), precio=VALUES(precio);

-- Alta de solicitud (estado inicial y fecha de orden)
INSERT INTO solicitud (estado, afiliado_dni, prestador_id, orden_fecha)
VALUES ('EN_EVALUACION', @af, @prest, CURDATE());
SET @sol := LAST_INSERT_ID();

-- Ítems: uno común y otro de alto monto
INSERT INTO item_solicitud (solicitud_id, prestacion_cod, cantidad, precio_unit)
VALUES
(@sol, 'P-001', 1, 12000.00),
(@sol, 'P-999', 1, 1200000.00);

-- Bitácora de alta
INSERT INTO bitacora (solicitud_id, actor, evento, detalle)
VALUES (@sol, 'administrativo', 'ALTA', 'Alta inicial de prueba');
```

```sql
1    select id, estado,orden_fecha from solicitud;
```

Result Grid | Filter Rows: | Edit: | Export

id	estado	orden_fecha
1	EN_EVALUACION	NULL
2	EN_EVALUACION	2025-09-25
NULL	NULL	NULL

Verificación de inserción en solicitud

```sql
1    select id, estado,orden_fecha from solicitud where id=@sol;
```

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

id	estado	orden_fecha
2	EN_EVALUACION	2025-09-25
NULL	NULL	NULL

Verificación por id

B) Consulta (estado, trazabilidad y totales)

Verifico el estado del expediente, la traza de bitácora (append-only) y el total mediante la vista.

```sql
-- Estado y fecha de orden
SELECT id, estado, orden_fecha
  FROM solicitud
 WHERE id = @sol;

-- Traza completa
SELECT fecha, actor, evento, detalle
  FROM bitacora
 WHERE solicitud_id = @sol
 ORDER BY fecha;

-- Items y total
SELECT i.prestacion_cod, i.cantidad, i.precio_unit
  FROM item_solicitud i
 WHERE i.solicitud_id = @sol;

SELECT *
  FROM vw_solicitud_total
 WHERE id = @sol;
```

```sql
SELECT fecha, actor, evento, detalle
  FROM bitacora
 WHERE solicitud_id = @sol
 ORDER BY fecha;
```

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

fecha	actor	evento	detalle
2025-09-25 09:42:50	administrativo	ALTA	Alta inicial de prueba

Bitácora de la solicitud

C) Duplicidad por ventana (control de antecedentes)

Dejo la consulta para validar duplicidad (ventana de 30 días). Si hace falta, inserto un antecedente de prueba.

```sql
INSERT INTO antecedentes (afiliado_dni, prestacion_cod, fecha)
VALUES (@af, 'P-001', DATE_SUB(CURDATE(), INTERVAL 10 DAY));

-- Consulta de duplicidad
SELECT COUNT(*) AS antecedentes_30d
  FROM antecedentes
 WHERE afiliado_dni = @af
   AND prestacion_cod = 'P-001'
   AND fecha >= DATE_SUB(CURDATE(), INTERVAL 30 DAY);
```

```sql
SELECT COUNT(*) AS antecedentes_30d
  FROM antecedentes
 WHERE afiliado_dni = @af
   AND prestacion_cod = 'P-001'
   AND fecha >= DATE_SUB(CURDATE(), INTERVAL 30 DAY);
```

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

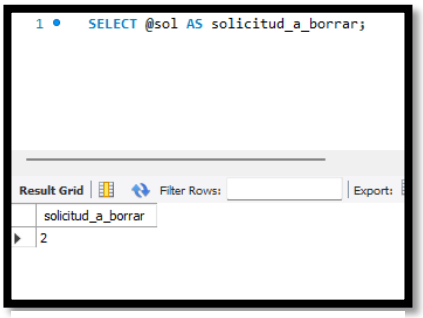
antecedentes_30d
1

Resultado de duplicidad (30 días)

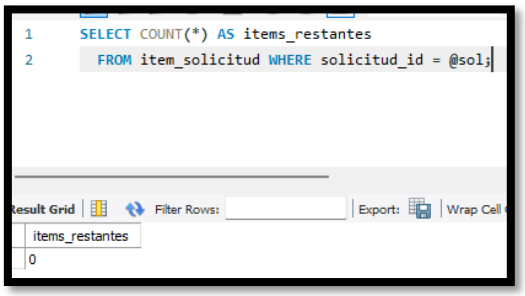
D) Borrado (solo limpieza de entorno de prueba)

En producción, los expedientes se **archivan**; el siguiente borrado es **exclusivo** para limpiar el entorno de testing. Gracias a **ON DELETE CASCADE**, al eliminar la solicitud se eliminan también sus ítems (y la bitácora **solo en pruebas**; en productivo sería **RESTRICT** para no perder auditoría).

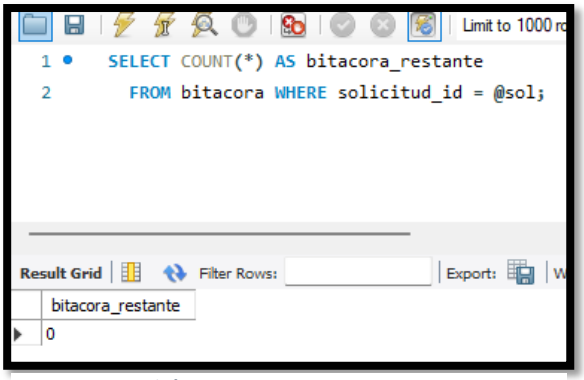
```
-- Ver que existe la solicitud
SELECT @sol AS solicitud_a_borrar;

-- Borrado del expediente de prueba
DELETE FROM solicitud WHERE id = @sol;

-- Verificación de cascada
SELECT COUNT(*) AS items_restantes
  FROM item_solicitud WHERE solicitud_id = @sol;

SELECT COUNT(*) AS bitacora_restante
  FROM bitacora WHERE solicitud_id = @sol;
```



Id a borrar



Cascada en ítems = 0



Cascada en bitácora = 0 (solo en pruebas)

Con estas operaciones demuestro que el esquema está **operativo**: puedo **insertar** altas con su **fecha de orden**, **consultar** estado y **trazabilidad** por bitácora, **calcular** el total por vista y **verificar** duplicidad por ventana. El **borrado** se usa solo para **limpieza de testing**; en el sistema final se seguirá la política de **archivado**.

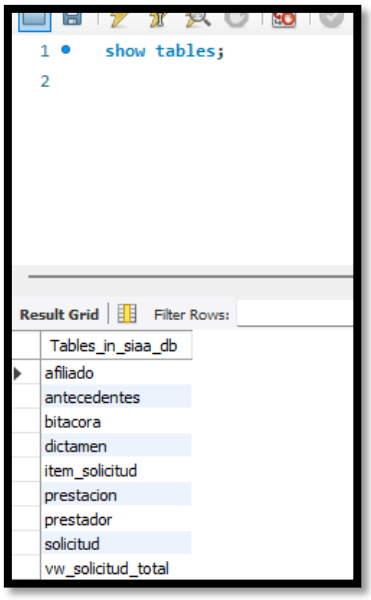
9. Presentación de las consultas SQL (evidencias)

Objetivo. Mostrar el conjunto de consultas que uso como evidencia del prototipo: estructura de la BD, estado de expedientes, trazabilidad por bitácora, totales por solicitud y control de duplicidad. Incluyo además un pequeño set de consultas **operativas** (resúmenes y métricas) para una vista mínima de gestión.

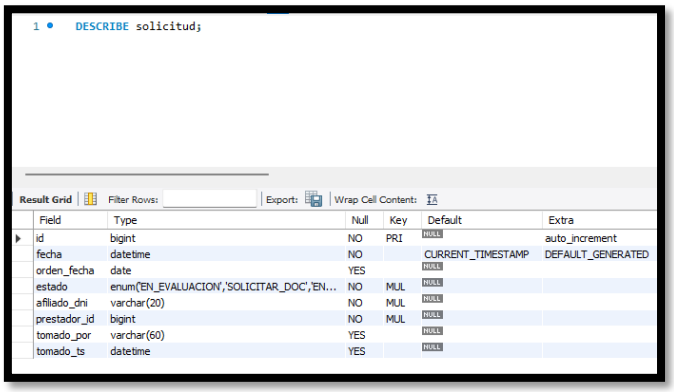
1) Consultas núcleo (evidencias del prototipo)

1.1 Estructura creada

```
SHOW TABLES;                                -- listado de tablas del esquema
DESCRIBE solicitud;                         -- estructura de la entidad principal
SHOW FULL TABLES WHERE Table_type = 'VIEW';-- verificar vistas
SHOW CREATE VIEW vw_solicitud_total;
```



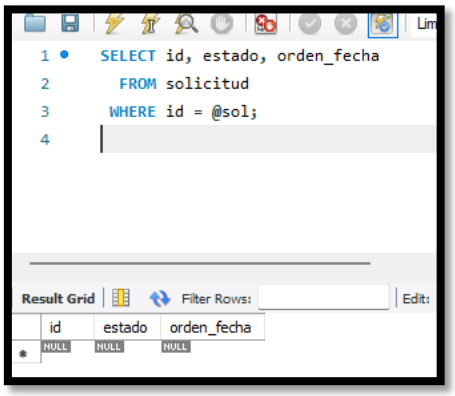
Salida de SHOW TABLES



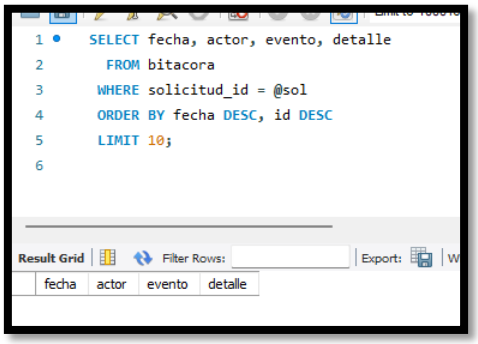
DESCRIBE solicitud

1.2 Estado y trazabilidad de un expediente

```
-- Estado actual (uso @sol seteado en la inserción de pruebas)
SELECT id, estado, orden_fecha
  FROM solicitud
 WHERE id = @sol;

-- Últimos eventos (append-only)
SELECT fecha, actor, evento, detalle
  FROM bitacora
 WHERE solicitud_id = @sol
 ORDER BY fecha DESC, id DESC
 LIMIT 10;
```



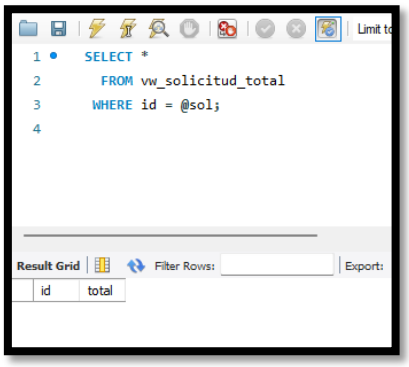
SELECT id, estado, orden_fecha



Bitácora ordenada

1.3 Totales por solicitud (vista)

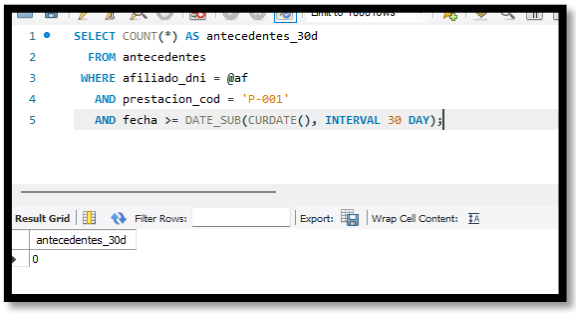
```
SELECT *
  FROM vw_solicitud_total
 WHERE id = @sol;
```



salida de la vista con el total de la solicitud

1.4 Control de duplicidad por ventana

```sql
-- Ventana de 30 días para una práctica puntual
SELECT COUNT(*) AS antecedentes_30d
  FROM antecedentes
 WHERE afiliado_dni = @af
   AND prestacion_cod = 'P-001'
   AND fecha >= DATE_SUB(CURDATE(), INTERVAL 30 DAY);
```



Resultado de duplicidad (0/1 según datos).

2) Consultas operativas (resúmenes y métricas mínimas) – *Opcional*

2.1 Autorizadas por prestador y mes

```sql
SELECT p.nombre AS prestador,
       DATE_FORMAT(s.fecha, '%Y-%m') AS periodo,
       COUNT(*) AS autorizadas
  FROM solicitud s
  JOIN prestador p ON p.id = s.prestador_id
 WHERE s.estado = 'AUTORIZADA'
 GROUP BY p.nombre, DATE_FORMAT(s.fecha, '%Y-%m')
 ORDER BY periodo DESC, autorizadas DESC;
```

2.2 Casos elevados y dictaminados

```sql
-- Elevadas pendientes de dictamen
SELECT s.id, s.fecha, s.afiliado_dni
  FROM solicitud s
  LEFT JOIN dictamen d ON d.solicitud_id = s.id
 WHERE s.estado = 'ELEVADA'
   AND d.id IS NULL;

-- Resumen de dictámenes
SELECT d.resultado, COUNT(*) AS cantidad
  FROM dictamen d
 GROUP BY d.resultado;
```

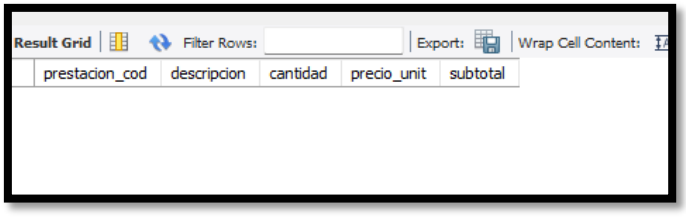
2.3 “Edad” del expediente (SLA simple)

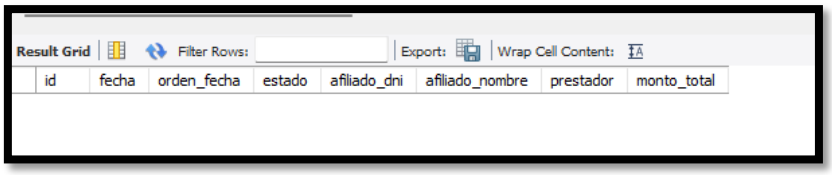
```sql
SELECT s.id, s.estado,
       TIMESTAMPDIFF(HOUR, s.fecha, NOW()) AS horas_transcurridas
  FROM solicitud s
 WHERE s.estado <> 'ARCHIVADA'
 ORDER BY horas_transcurridas DESC
 LIMIT 20;
```

3) Ficha rápida del expediente (consolidado)

Una consulta “todo en uno” para ver encabezado, ítems, total y último evento de bitácora.

```sql
-- Último evento de bitácora (subconsulta)
SELECT b.*
  FROM bitacora b
 WHERE b.solicitud_id = @sol
 ORDER BY b.fecha DESC, b.id DESC
 LIMIT 1;

-- Encabezado + total
SELECT s.id, s.fecha, s.orden_fecha, s.estado,
       a.dni AS afiliado_dni, a.nombre AS afiliado_nombre,
       p.nombre AS prestador,
       v.total AS monto_total
  FROM solicitud s
  JOIN afiliado a ON a.dni = s.afiliado_dni
  JOIN prestador p ON p.id = s.prestador_id
  LEFT JOIN vw_solicitud_total v ON v.id = s.id
 WHERE s.id = @sol;

-- Ítems
SELECT i.prestacion_cod, pr.descripcion, i.cantidad, i.precio_unit,
       (i.cantidad * i.precio_unit) AS subtotal
  FROM item_solicitud i
  JOIN prestacion pr ON pr.codigo = i.prestacion_cod
 WHERE i.solicitud_id = @sol
 ORDER BY pr.descripcion;
```





10. Definiciones de comunicación

En este punto explico cómo se establece la comunicación entre el sistema SIAA y la base de datos MySQL, además de las condiciones operativas y de seguridad que acompañan a dicha conexión. El objetivo es documentar de forma clara los protocolos, parámetros y escenarios de despliegue previstos, asegurando la trazabilidad con lo definido en el diseño (sección 2.4).

10.1 Escenarios de despliegue

Para el prototipo se contemplan dos formas de operación:

- **Escenario A – Local:** la aplicación Java y MySQL residen en la misma PC. Este esquema resulta útil para desarrollo y pruebas iniciales, ya que reduce la complejidad de red.
- **Escenario B – Cliente/Servidor:** la aplicación se ejecuta en los puestos (Administrativo y Médico) y se conecta a un servidor de base de datos a través de la red local.

En ambos casos, la comunicación se realiza mediante **JDBC sobre TCP/3306**. En el escenario cliente/servidor, el firewall se configura para permitir dicho puerto exclusivamente a la subred de los puestos habilitados.

10.2 Protocolo y parámetros de conexión

La conexión se implementa con **JDBC** y el conector oficial de MySQL. La URL base de conexión es:

```
jdbc:mysql://<anfitrión>:3306/siaa_db
    ?useSSL=true
    &requireSSL=true
    &characterEncoding=utf8
    &serverTimezone=UTC
```

De esta forma, la comunicación se establece cifrada (TLS), con codificación UTF-8 y zona horaria fija para evitar inconsistencias en los cálculos de ventana temporal.

Los parámetros de conexión se externalizan en un archivo *config.properties*, lo que permite cambiar host, puerto, esquema o credenciales sin modificar el código:

```
Db.anfitrión=192.168.10.20
Db.puerto=3306
db.schema=siaa_db
Db.usuario=app_siaa
db.pasar=********
Db.SSL=verdadero
```

10.3 Seguridad y gestión de credenciales

Para reducir riesgos se sigue el principio de **mínimo privilegio**. El usuario app_siaa solo dispone de permisos de lectura y escritura sobre las tablas necesarias, sin facultad de

crear o eliminar estructuras. Particularmente, la tabla **bitácora** se define como *append-only*: permite inserciones y consultas, pero no actualizaciones ni eliminaciones.

Además, las credenciales se almacenan únicamente en archivos de configuración externos y no se incluyen en repositorios de código, con la posibilidad de rotarlas periódicamente.

10.4 Consideraciones operativas

El sistema abre y cierra conexiones mediante JDBC de forma controlada, agrupando cada decisión (alta, rechazo, autorización, anulación) en una transacción que asegura atomicidad y consistencia. En caso de error, se aplica rollback y se deja registro en la bitácora.

Las consultas se implementan con PreparedStatement para prevenir inyecciones SQL, y se definen tiempos de espera (timeouts) razonables para detectar fallas de red. Los respaldos de la base de datos se programan desde el servidor MySQL, complementados con políticas de firewall que limitan el acceso al puerto 3306 únicamente a los puestos autorizados.

10.5 Relación con el diseño

El modelo de comunicación documentado se corresponde con el **diagrama de despliegue** presentado en el diseño (2.4). En este se representan los artefactos (cliente Java y archivo de configuración), los nodos (PC única o red con servidor de base de datos) y los canales (JDBC/TCP/3306). También se reflejan las reglas operativas: usuario con privilegios mínimos, firewall y backups programados.

Bibliografía

- Kendall, K. & Kendall, J. (2011). *Análisis y diseño de sistemas*. Pearson Education.
- Sommerville, I. (2011). *Ingeniería del Software* (9ª ed.). Pearson Educación.
- Pressman, R. (2010). *Ingeniería del Software: un enfoque práctico* (7ª ed.). McGraw-Hill.
- Universidad Siglo 21. (2025). *Seminario de Práctica – Módulo 2: Modelado, Diseño e Implementación*. Material de cátedra.
- Oracle. (2024). *MySQL 8.0 Reference Manual*. Recuperado de https://dev.mysql.com/doc/
- Oracle. (2024). *Java Platform, Standard Edition Documentation*. Recuperado de https://docs.oracle.com/en/java/
- GitHub Docs. (2025). *Managing repositories*. Recuperado de https://docs.github.com