

Universidad de Las Américas
Facultad de Ingenierías y Ciencias Agropecuarias
Ingeniería De Software
Progreso 1

Nombres: Enrique Merizalde

Fecha: 23/04/2025

INFORME PRUEBA DE INTEGRACIÓN

Identificación del Problema:

Tras analizar el estudio de la empresa BioNet, se han identificado varios problemas críticos en la transferencia de información de los resultados de los exámenes clínicos realizados a los pacientes en sus diferentes sucursales. El principal desafío radica en el proceso manual de transferencia de archivos, lo cual introduce una alta susceptibilidad al error humano. Este método de transferencia es propenso a fallos como la omisión de archivos, la introducción de datos incorrectos o la sobreescritura involuntaria de información, lo que pone en riesgo la integridad y precisión de los resultados.

Además, el sistema central accede a una base de datos compartida que concentra toda la información, lo que crea varios puntos de vulnerabilidad. La concurrencia de múltiples procesos que intentan escribir en la misma base de datos de manera simultánea puede generar conflictos de escritura y problemas de sincronización. Esto puede resultar en datos incompletos, inconsistentes o desactualizados, afectando la calidad del servicio y la fiabilidad de los registros de los exámenes. El uso de una base de datos compartida sin un control adecuado también aumenta la posibilidad de que se produzcan duplicados o pérdida de datos, lo que compromete la precisión y la integridad de la información consolidada.

Justificación de uso de los patrones de transferencia de archivos y BD compartida:

Una solución efectiva para abordar los problemas actuales es la implementación de una base de datos compartida centralizada, donde toda la información de los exámenes sea consolidada de manera única. Al centralizar la data en una única base de datos, se garantiza la consistencia y se evita la duplicación de registros, lo que no solo mejora la calidad de los datos, sino también facilita el control y mantenimiento de la información. La gestión de duplicados y la validación de datos se puede realizar de manera eficiente mediante procedimientos almacenados en SQL o lógica de validación implementada en el código de la aplicación, asegurando que solo datos válidos y únicos sean ingresados.

Por otro lado, al utilizar el patrón de transferencia de archivos, se elimina la dependencia de procesos manuales que son propensos a errores humanos. Con este enfoque, cuando un archivo .CSV es generado por cualquier laboratorio, este será automáticamente procesado, validado y movido a su destino correspondiente, ya sea la base de datos o la carpeta de errores en caso de que el archivo no cumpla con los requisitos esperados. Esto asegura un flujo de trabajo más eficiente, rápido y menos susceptible a fallos operativos. Al automatizar el proceso de transferencia, se minimizan las intervenciones humanas y se mejora la fiabilidad del sistema en su conjunto.

Diseño a Alto Nivel de la Solución:

La solución propuesta tiene como objetivo automatizar los procesos clave de revisión y categorización de los archivos .csv, eliminando la intervención manual y los riesgos asociados a errores humanos. A alto nivel, el flujo de trabajo se organiza de la siguiente manera:

1. **Revisión y Validación Automática de Archivos:** Cuando un archivo .csv es generado en cualquiera de las sucursales, se coloca automáticamente en la carpeta de entrada /input-labs. Desde allí, un proceso automatizado se encarga de revisar y validar el archivo según criterios predefinidos (como la estructura, los campos requeridos y la integridad de los datos). Este proceso asegura que solo los archivos correctamente formateados y completos pasen al siguiente paso, sin intervención humana.

2. **Categorización de Archivos:** Los archivos que cumplen con los criterios de validación son movidos a la carpeta **/processed**, mientras que aquellos que no cumplen con los requisitos (por ejemplo, archivos incompletos o mal formateados) se mueven a la carpeta **/error** para ser revisados y corregidos por el personal encargado.
3. **Inserción de Datos en la Base de Datos:** Los archivos que se encuentran en la carpeta **/processed** son automáticamente procesados e insertados en la base de datos central. Antes de la inserción, se lleva a cabo un procedimiento de validación para verificar que no existan duplicados, asegurando que los datos sean únicos y no se sobrescriban. Este control de duplicados se realiza mediante la aplicación de reglas de negocio definidas en el código o en consultas SQL específicas.
4. **Registro de Cambios en la Base de Datos:** Cada modificación en la base de datos, ya sea una nueva inserción o una actualización, es registrada en un log de auditoría. Este log permite un seguimiento detallado de los cambios realizados, lo que facilita la trazabilidad y el control de la información.

Este enfoque automatizado no solo mejora la eficiencia y la precisión del proceso, sino que también minimiza el riesgo de errores humanos y facilita la gestión de los datos a lo largo de todo el flujo de trabajo.

Estructura de carpetas

```
PS C:\INTEGRACION\PruebaP1> tree
Folder PATH listing for volume Windows
Volume serial number is 80DE-5952
C:.\
├── .mvn
│   └── wrapper
├── error
├── input-labs
├── processed
├── src
│   ├── main
│   │   ├── java
│   │   │   ├── com
│   │   │   │   ├── example
│   │   │   │   │   ├── demo
│   │   │   │   │   └── model
│   │   └── resources
│   └── test
│       ├── java
│       │   ├── com
│       │   │   ├── example
│       │   │   └── demo
│       └── target
│           ├── classes
│           │   ├── com
│           │   │   ├── example
│           │   │   │   ├── demo
│           │   │   │   └── model
│           ├── generated-sources
│           │   └── annotations
│           ├── generated-test-sources
│           │   └── test-annotations
│           ├── maven-archiver
│           ├── maven-status
│           │   ├── maven-compiler-plugin
│           │   │   ├── compile
│           │   │   │   ├── default-compile
│           │   │   └── testCompile
│           │   │       └── default-testCompile
│           └── test-classes
│               ├── com
│               │   ├── example
│               │   └── demo
└── target
```

PS C:\INTEGRACION\PruebaP1>

Flujo de Integración:

El flujo de integración diseñado para la solución se basa en la automatización de la transferencia, validación y consolidación de archivos .csv provenientes de distintas sucursales de BioNet. Este proceso se desarrolla en dos módulos principales: la transferencia y categorización de archivos, y la ingesta de datos en una base de datos compartida.

1. Transferencia y Categorización de Archivos (.csv):

Para automatizar este proceso, se ha utilizado Apache Camel, una herramienta de integración liviana que permite definir rutas de procesamiento de archivos. Se ha configurado un proyecto en Spring Boot con Maven, incorporando la dependencia de Camel para monitorear constantemente una carpeta de entrada (/input-labs).

Cada vez que un archivo .csv es detectado en esta carpeta, Camel lee su contenido y verifica la validez del archivo en función de su cabecera. Para este caso, un archivo se considera válido si contiene exactamente los siguientes campos en el encabezado:

laboratorio_id, paciente_id, tipo_examen, resultado, fecha_examen.

- Si el archivo cumple con esta estructura, es movido automáticamente a la carpeta /processed.
- Si el archivo es inválido (por campos incompletos, mal estructurados o corruptos), se traslada a la carpeta /error, facilitando su revisión posterior por el equipo correspondiente.

2. Base de Datos Compartida (SQL Server):

Se ha implementado una base de datos local utilizando SQL Server Management Studio, que incluye dos tablas:

- resultados_examenes: Contiene los datos consolidados de los exámenes con los campos:
id, laboratorio_id, paciente_id, tipo_examen, resultado, fecha_examen.
Para evitar la duplicación de registros, se ha definido una restricción UNIQUE sobre las columnas:
(laboratorio_id, paciente_id, tipo_examen, fecha_examen).
- log_cambios_resultados: Almacena el historial de inserciones y modificaciones realizadas sobre la tabla de resultados. Sus campos

son:

id, operacion, paciente_id, tipo_examen, fecha.

Esta tabla es gestionada mediante un trigger que se activa cada vez que ocurre un INSERT o UPDATE en resultados_exámenes, registrando la operación correspondiente.

3. Lógica de Inserción desde Archivos Procesados:

Finalmente, se ha implementado la lógica en **Java** que permite que, una vez que un archivo válido ha sido clasificado en la carpeta **/processed**, se procese línea por línea e inserte los datos en la base de datos compartida. Antes de insertar, se verifica automáticamente que no existan duplicados, en cumplimiento con la restricción UNIQUE definida, asegurando la integridad de los datos.

Este flujo garantiza una integración segura y controlada entre los sistemas distribuidos de los laboratorios y el sistema central, minimizando errores humanos, eliminando duplicados y dejando trazabilidad de todos los cambios realizados.

Esquema de base de datos sugerido (Query)

```
CREATE DATABASE laboratorio
GO
```

```
-- Crear Usuario para base --
```

```
CREATE USER emeri FOR LOGIN emeri;
EXEC sp_addrolemember 'db_datareader', emeri; -- Read permissions
EXEC sp_addrolemember 'db_datawriter', emeri; -- Write permissions
```

```
USE laboratorio
GO
```

```
CREATE TABLE resultados_exámenes (
    id INT IDENTITY (1,1) PRIMARY KEY,
    laboratorio_id INT NOT NULL,
    paciente_id INT NOT NULL,
    tipo_examen VARCHAR(255) NOT NULL,
    resultado TEXT NOT NULL,
```

```
    fecha_examen DATETIME NOT NULL,  
    UNIQUE (laboratorio_id, paciente_id, tipo_examen, fecha_examen) -- Evitar  
    duplicados  
);
```

```
CREATE TABLE log_cambios_resultados (  
    id INT IDENTITY (1,1) PRIMARY KEY,  
    operacion VARCHAR(50) NOT NULL,  
    paciente_id INT NOT NULL,  
    tipo_examen VARCHAR(255) NOT NULL,  
    fecha DATETIME NOT NULL  
);
```

```
DROP TRIGGER after_resultados_insert
```

```
CREATE TRIGGER after_resultados_insert  
ON resultados_examenes  
AFTER INSERT, UPDATE  
AS  
BEGIN  
    -- Para INSERT  
    INSERT INTO log_cambios_resultados (operacion, paciente_id,  
    tipo_examen, fecha)  
    SELECT 'INSERT', paciente_id, tipo_examen, GETDATE()  
    FROM inserted;  
  
    -- Para UPDATE  
    IF EXISTS (SELECT * FROM inserted)  
    AND EXISTS (SELECT * FROM deleted)  
    BEGIN  
        INSERT INTO log_cambios_resultados (operacion, paciente_id,  
    tipo_examen, fecha)  
        SELECT 'UPDATE', paciente_id, tipo_examen, GETDATE()  
        FROM inserted;  
    END  
END;
```

```
-- Test  
INSERT INTO resultados_examenes (laboratorio_id, paciente_id, tipo_examen,  
resultado, fecha_examen)  
VALUES (1, 101, 'Hemograma', 'Normal', '2025-04-23');
```

```
SELECT * FROM resultados_examenes
```

```
UPDATE resultados_examenes  
SET resultado = 'Anormal'  
WHERE paciente_id = 101 AND tipo_examen = 'Hemograma';
```

```
SELECT * FROM resultados_examenes
```

```
SELECT * FROM log_cambios_resultados;
```

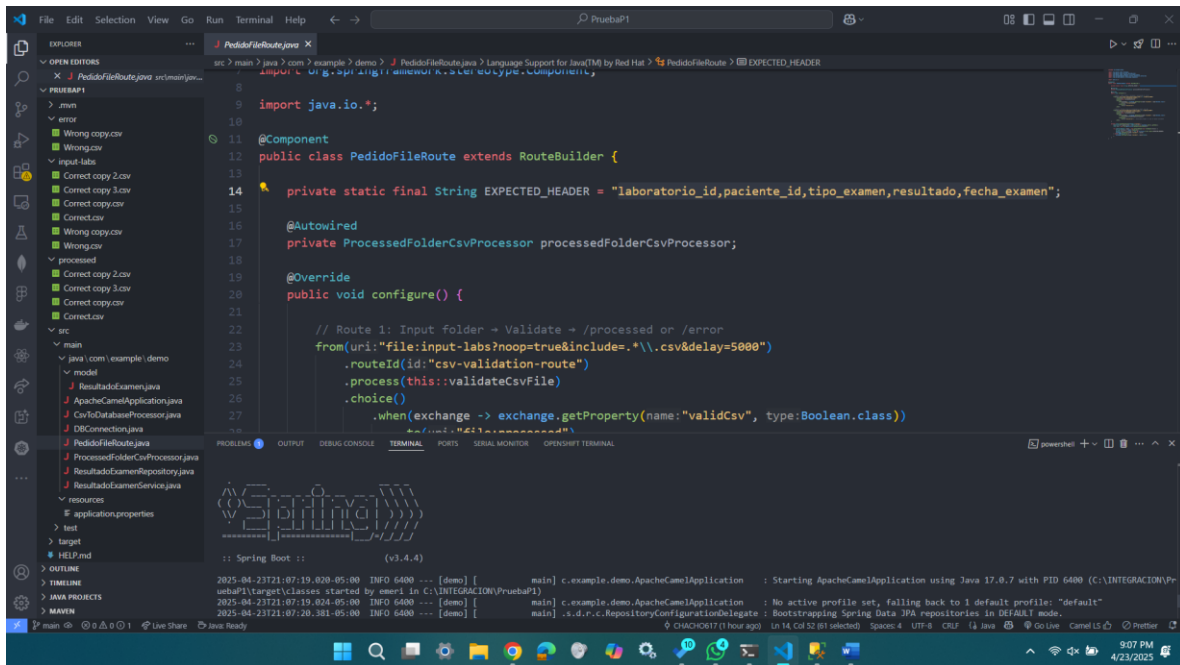
Esquema de base de datos sugerido (Grafico)

resultados examenes	
🔑	id
	laboratorio_id
	paciente_id
	tipo_examen
	resultado
	fecha_examen

log cambios resultados	
🔑	id
	operacion
	paciente_id
	tipo_examen
	fecha

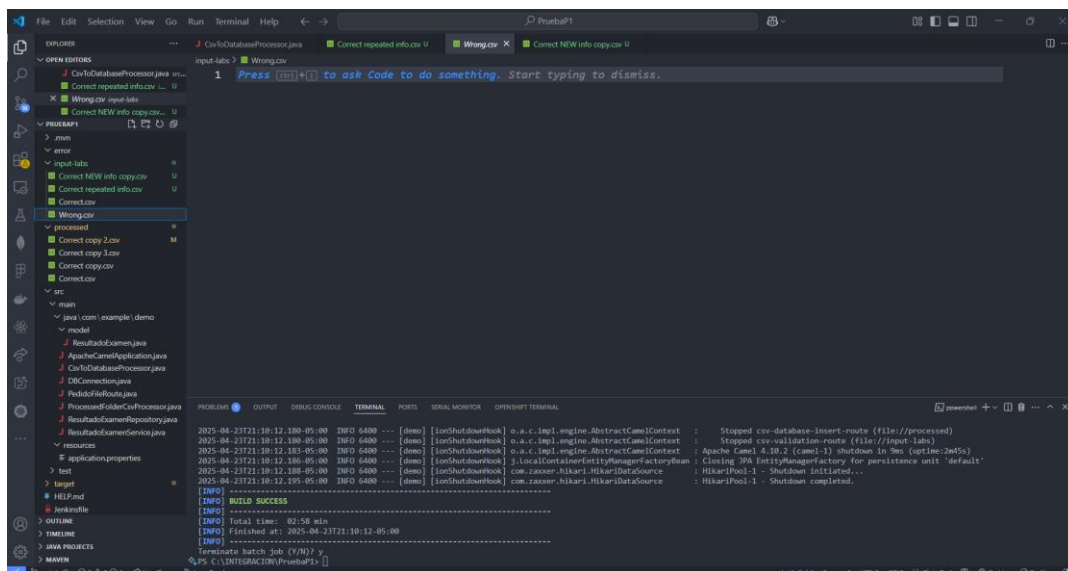
Implementación:

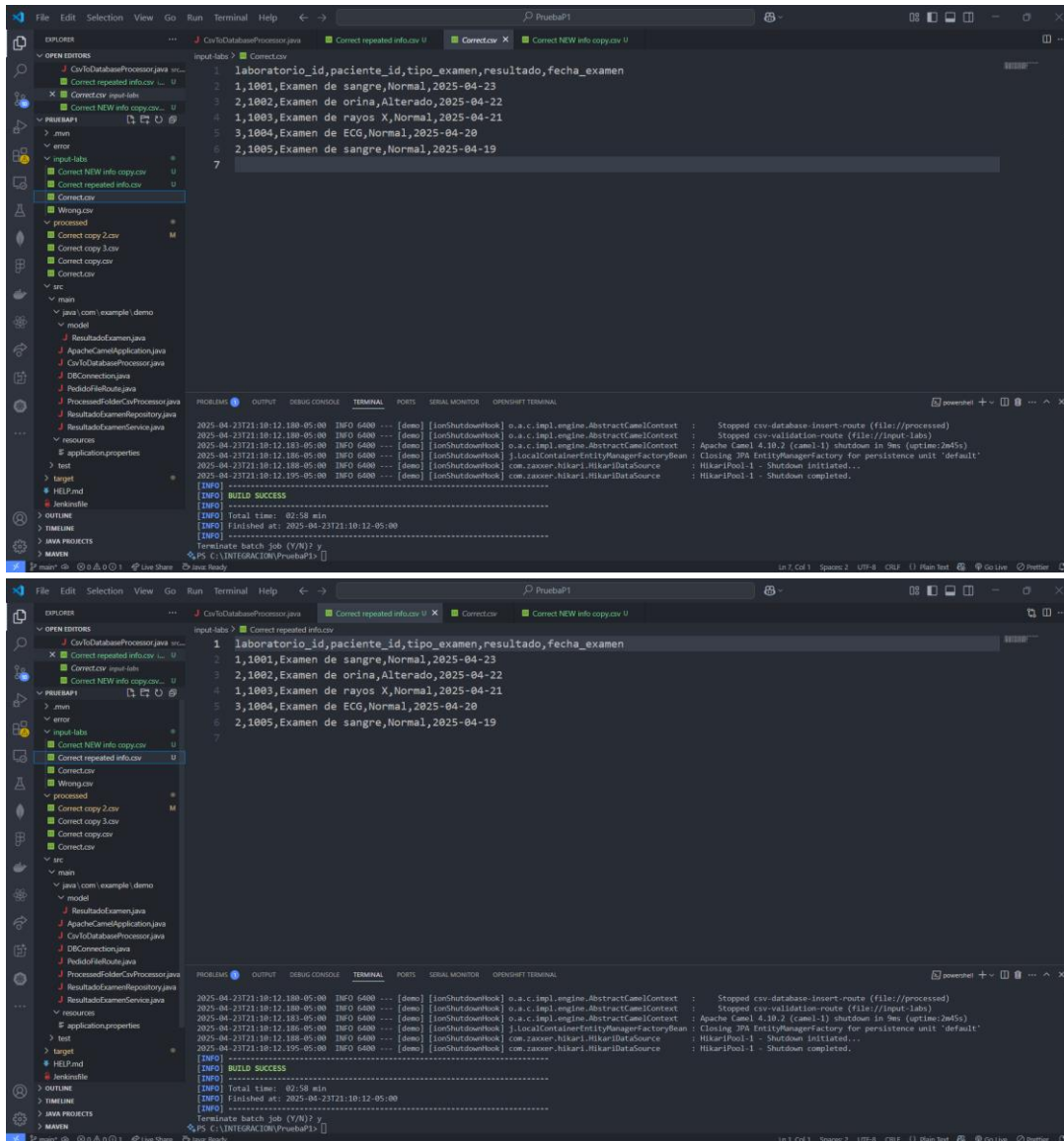
https://github.com/CHACHO617/Integracion_P1



Pruebas de funcionamiento:

1. Se han creado 4 csvs, uno con falla, uno con informacion duplicada, uno con informacion vaida y uno con informacion nueva.





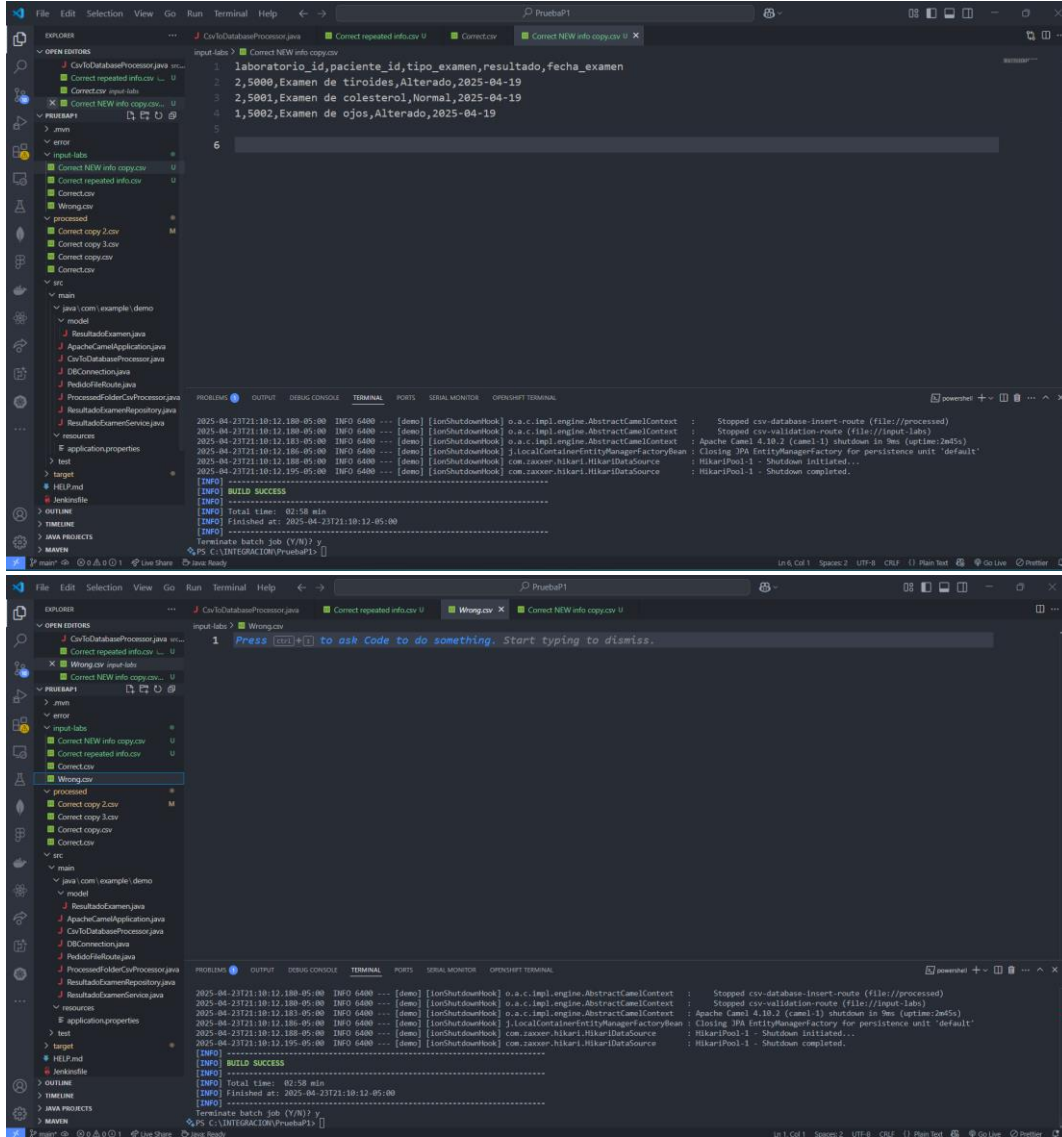
The screenshot shows an IDE with a project named 'PruebaP1'. The project structure includes a 'main' directory with 'input-labs' and 'resources' subdirectories. The 'input-labs' directory contains several CSV files: 'Correct.csv', 'Wrong.csv', 'Correct copy 2.csv', 'Correct copy 3.csv', 'Correct.csv', 'Correct NEW info copy.csv', 'Correct repeated info.csv', and 'Correct NEW info copy.csv'. The 'resources' directory contains 'application.properties'. The 'main' directory also contains 'ResultadoExamen.java', 'ApacheCamelApplication.java', 'CsvToDatabaseProcessor.java', 'DBConnection.java', 'PedidoFactory.java', 'ProcessadorCsvProcessor.java', 'ResultadoExamenRepository.java', 'ResultadoExamenService.java', and 'resources'. The 'test' directory contains 'application.properties', 'target', 'HELP.md', and 'testutils'. The 'Maven' directory contains 'pom.xml'.

The terminal window displays the following logs:

```

2025-04-23T21:10:12.180-05:00 INFO 6480 --- [demo] [ioShutdownHook] o.a.c.impl.engine.AbstractCamelContext : Stopped csv-database-insert-route (file://processed)
2025-04-23T21:10:12.180-05:00 INFO 6480 --- [demo] [ioShutdownHook] o.a.c.impl.engine.AbstractCamelContext : Stopped csv-validation-route (file://input-labs)
2025-04-23T21:10:12.183-05:00 INFO 6480 --- [demo] [ioShutdownHook] o.a.c.impl.engine.AbstractCamelContext : Apache Camel 4.10.2 (camel-1) shutdown in 9ms (uptime:2m45s)
2025-04-23T21:10:12.186-05:00 INFO 6480 --- [demo] [ioShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2025-04-23T21:10:12.188-05:00 INFO 6480 --- [demo] [ioShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2025-04-23T21:10:12.195-05:00 INFO 6480 --- [demo] [ioShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.

[INFO] BUILD SUCCESS
[INFO] Total time: 02:58 min
[INFO] Finished at: 2025-04-23T21:10:12-05:00
[INFO]
Terminate batch job (Y/N)? y
  
```



The image displays two screenshots of an IDE (IntelliJ IDEA) showing a Java project with a Camel route. The top screenshot shows the route definition with a 'Correct NEW info copy.csv' processor. The bottom screenshot shows the same route with a 'Wrong.csv' processor, and the terminal output shows the route running successfully.

Top Screenshot: The IDE shows a project named 'Prueba1'. The 'Correct NEW info copy.csv' processor is defined in the route. The terminal output shows the route running successfully, with logs indicating the route is started and the processor is executed.

```

1 laboratorio_id,paciente_id,tipo_examen,resultado,fecha_examen
2 2,5000,Examen de tiroides,Alterado,2025-04-19
3 2,5001,Examen de colesterol,Normal,2025-04-19
4 1,5002,Examen de ojos,Alterado,2025-04-19
5
6

```

Bottom Screenshot: The IDE shows the same project, but the 'Wrong.csv' processor is now selected. The terminal output shows the route running successfully, with logs indicating the route is started and the processor is executed.

```

1 Press [Ctrl]+[Z] to ask Code to do something. Start typing to dismiss.

```

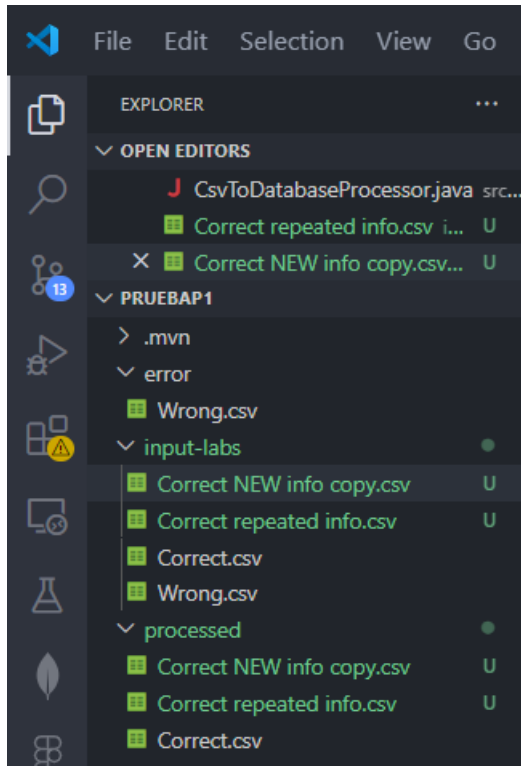
The terminal output for both screenshots is as follows:

```

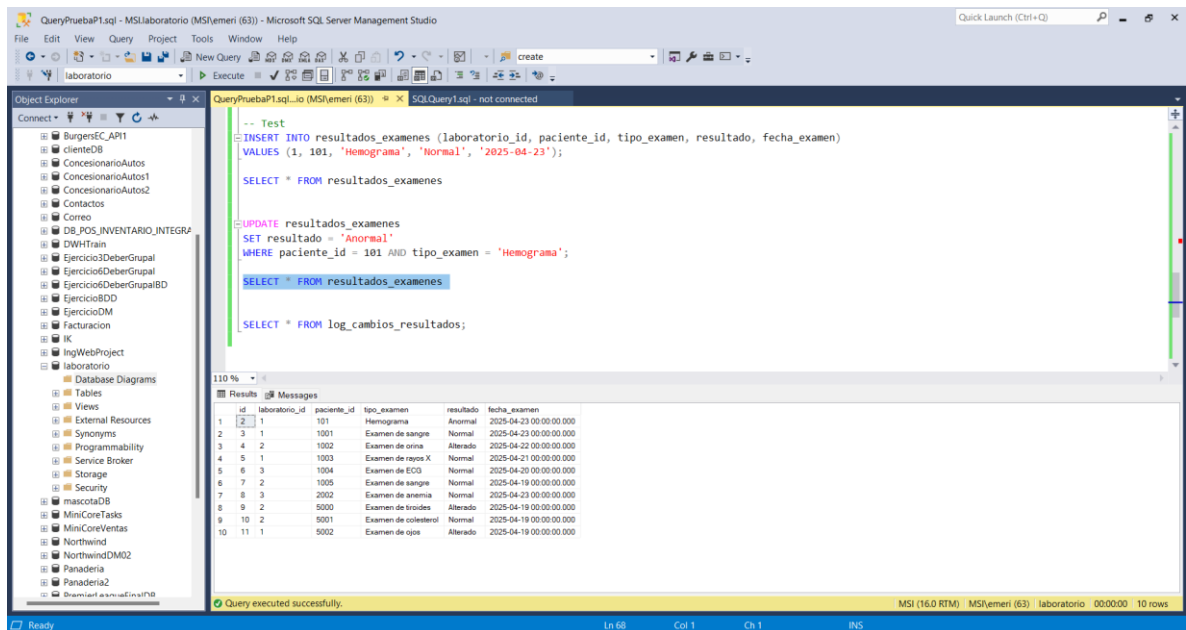
2025-04-23T21:10:12.180-05:00 INFO 6400 --- [demo] [onShutdownHook] o.a.c.impl.engine.AbstractCamelContext : Stopped csv-database-insert-route (file://processed)
2025-04-23T21:10:12.180-05:00 INFO 6400 --- [demo] [onShutdownHook] o.a.c.impl.engine.AbstractCamelContext : Stopped csv-validation-route (file://input-labs)
2025-04-23T21:10:12.183-05:00 INFO 6400 --- [demo] [onShutdownHook] o.a.c.impl.engine.AbstractCamelContext : Apache Camel 4.10.2 (camel-1) shutdown in 9ms (uptime:2m5s)
2025-04-23T21:10:12.186-05:00 INFO 6400 --- [demo] [onShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2025-04-23T21:10:12.186-05:00 INFO 6400 --- [demo] [onShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2025-04-23T21:10:12.195-05:00 INFO 6400 --- [demo] [onShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
[INFO] BUILD SUCCESS
[INFO] Total time: 02:58 min
[INFO] Finished at: 2025-04-23T21:10:12-05:00
[INFO]
Terminate batch job (Y/N)? y
PS C:\INTEGRACION\Prueba1>

```

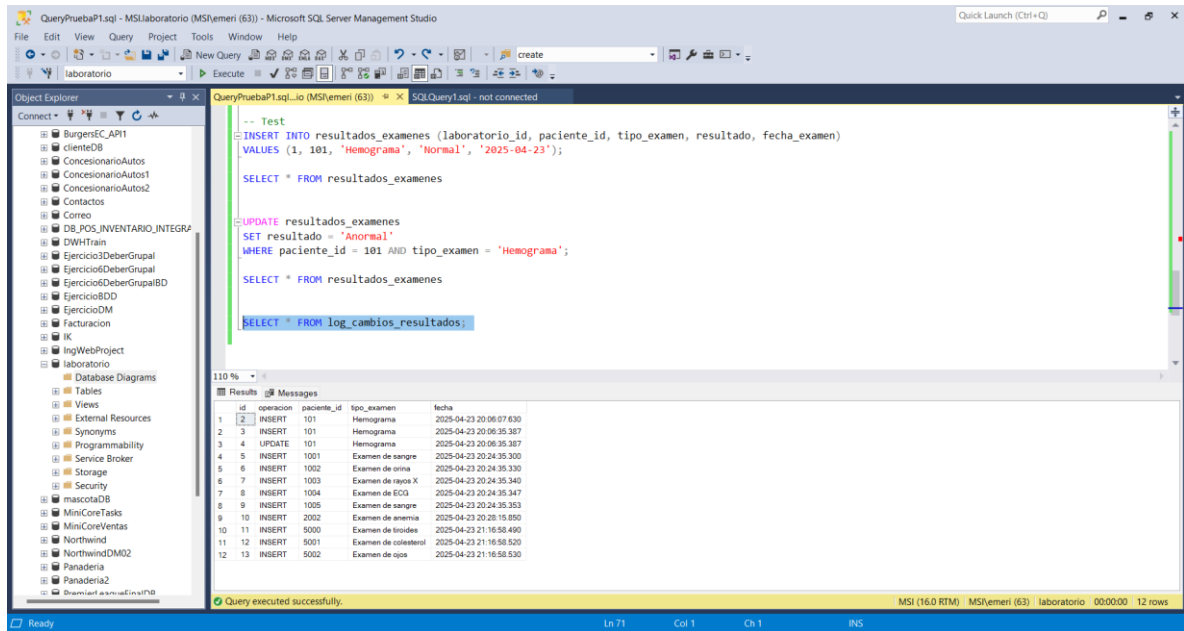
2. Prueba de que se mueven los datos



3. Insertar a la base de datos sin duplicados



4. Log de cambios



The screenshot shows the Microsoft SQL Server Enterprise Manager interface. The left pane displays the 'Object Explorer' with a tree view of the 'laboratorio' database. The central pane shows a SQL query window with the following code:

```
-- Test
INSERT INTO resultados_examen (laboratorio_id, paciente_id, tipo_examen, resultado, fecha_examen)
VALUES (1, 101, 'Hemograma', 'Normal', '2025-04-23');

SELECT * FROM resultados_examen

UPDATE resultados_examen
SET resultado = 'Anormal'
WHERE paciente_id = 101 AND tipo_examen = 'Hemograma';

SELECT * FROM resultados_examen

SELECT * FROM log_cambios_resultados;
```

The bottom pane shows the 'Results' window with the following data:

id	operacion	paciente_id	tipo_examen	fecha
1	2	101	Hemograma	2025-04-23 20:06:07.630
2	3	101	Hemograma	2025-04-23 20:06:35.387
3	4	101	Hemograma	2025-04-23 20:06:35.387
4	5	1001	Examen de sangre	2025-04-23 20:24:35.300
5	6	1002	Examen de orina	2025-04-23 20:24:35.330
6	7	1003	Examen de rayos X	2025-04-23 20:24:35.340
7	8	1004	Examen de ECG	2025-04-23 20:24:35.347
8	9	1005	Examen de sangre	2025-04-23 20:24:35.353
9	10	2002	Examen de anemia	2025-04-23 20:28:15.850
10	11	5000	Examen de triodes	2025-04-23 21:16:58.490
11	12	5001	Examen de colesterol	2025-04-23 21:16:58.520
12	13	5002	Examen de ojos	2025-04-23 21:16:58.530

The status bar at the bottom indicates 'Query executed successfully.' and '12 rows'.