Paula De León – 23202

Adrián González – 23152

Análisis de Laboratorio 3

1. ¿Por qué eligieron ese ORM y qué beneficios o dificultades encontraron?

Utilizamos Eloquent, el ORM nativo de Laravel, junto con PostgreSQL como motor de base de datos. La razón principal fue su integración fluida con Laravel, lo que nos permitió desarrollar más rápido y de forma más organizada.

2. ¿Cómo implementaron la lógica master-detail dentro del mismo formulario?

Se usó un formulario donde el "master" es el estudiante y los "details" son sus inscripciones a cursos.
Para lograrlo, la información se guarda de forma transaccional, primero se valida y guarda el estudiante, luego las inscripciones relacionadas. La lógica de validación se aplica en cascada, asegurando que los datos estén completos antes de grabarlos en la base de datos.

3. ¿Qué validaciones implementaron en la base de datos y cuáles en el código?

En la base de datos:

- NOT NULL en campos obligatorios.
- UNIQUE para evitar duplicados (como estudiante-curso).
- CHECK para validar rangos (fechas, créditos, etc.).
- ENUM para estados controlados (como estado_inscripcion).

En el código (Laravel):

- Validación de formatos (email, date, etc.).
- Reglas de negocio como disponibilidad de cupo.
- Validación condicional según contexto.

4. ¿Qué beneficios encontraron al usar tipos de datos personalizados?

Utilizar tipos como ENUM nos ayudó a validar valores directamente desde la base de datos, mejorar la legibilidad del código, evitar errores por datos mal ingresados y centralizar los valores permitidos, facilitando cambios futuros. Además, nos permitió documentar mejor el sistema (los valores posibles están claros desde el esquema).

5. ¿Qué ventajas ofrece usar una VIEW como base del índice en vez de una consulta directa?

La vista vista_estudiantes_cursos centraliza la lógica de varias tablas y ofrece muchas ventajas como:

- **Abstracción**: Oculta la complejidad de los JOIN.
- **Reutilización**: Puede usarse en reportes, APIs y dashboards.
- **Consistencia**: Siempre representa la misma lógica en toda la aplicación.
- **Mantenibilidad**: Cambios en una sola vista afectan todas sus instancias.
- **Optimización**: Puede indexarse o materializarse para mejorar el rendimiento.

6. ¿Qué escenarios podrían romper la lógica actual si no existieran las restricciones?

Sin las restricciones que colocamos podrían ocurrir errores como tan simples como estudiantes duplicados en el mismo curso, cursos sin profesor asignado, fechas o edades inválidas, calificaciones fuera de rango, eliminación de registros que dejan huérfanas otras tablas o incluso inscripciones a cursos inexistentes.

7. ¿Qué aprendieron sobre la separación entre lógica de aplicación y lógica de persistencia?

Aprendimos que cada capa tiene su rol. Por ejemplo, la base de datos debe garantizar la integridad mínima, la aplicación maneja la lógica de negocio más compleja, así como las validaciones y procesos deben de estar desacoplados para facilitar mantenimiento y pruebas. Las transacciones ayudan a mantener la coherencia entre ambas capas y las vistas pueden actuar como interfaz entre backend y reportes o APIs.

8. ¿Cómo escalaría este diseño en una base de datos de gran tamaño?

A gran escala, el sistema debería adaptarse mediante índices en columnas clave para búsquedas frecuentes, paginación estricta en interfaces de usuario y organización de tablas por fechas o regiones. También podría haber uso de vistas materializadas para reportes pesados y réplicas de lectura para mejorar rendimiento.

9. ¿Consideran que este diseño es adecuado para una arquitectura con microservicios?

Actualmente, el sistema está más orientado a una arquitectura monolítica, pero con algunos ajustes podría dividirse en microservicios, por ejemplo, un servicio enfocado en estudiantes, otro en cursos y otro en inscripciones. Lo más desafiante podrían ser las relaciones estrechas, transacciones distribuidas, sincronización de datos.

10. ¿Cómo reutilizarían la vista en otros contextos como reportes o APIs?

La vista vista_estudiantes_cursos puede utilizarse para:

- **Exportación de reportes** (Excel, PDF).
- **Dashboards** con métricas clave.
- **APIs** que devuelvan información consolidada.
- **Filtros avanzados** para búsquedas complejas.

11. ¿Qué decisiones tomaron para estructurar su modelo de datos y por qué?

Se normalizaron las tablas hasta tercera forma normal (3FN) para evitar redundancia. También se incluyó una tabla intermedia (estudiante_curso) con atributos adicionales como notas o fecha de inscripción y se usaron timestamps para auditoría. Se agregaron índices en campos de búsqueda frecuente, se optó por claves foráneas y nombres descriptivos para facilitar el mantenimiento.

12. ¿Cómo documentaron su modelo para facilitar su comprensión por otros desarrolladores?

Se utilizaron diagramas entidad-relación (ERD), comentarios en migraciones y seeders, convenciones claras en nombres de tablas y columnas. Se añadieron ejemplos de datos en archivos seeder o scripts SQL y documentación de relaciones en los modelos Eloquent

13. ¿Cómo evitaron la duplicación de registros o errores de asignación en la tabla intermedia?

Pusimos una restricción UNIQUE en los campos (id_estudiante, id_curso) en la tabla intermedia, hubo validaciones desde el backend para evitar registros duplicados y uso de transacciones para operaciones múltiples (guardar estudiante + inscripciones). Consideramos la posibilidad de usar sync() de Eloquent para mantener las relaciones de forma controlada. En el futuro podrían agregarse triggers o bloqueos optimistas si la concurrencia lo requiere.