Informe Reto 3: Que aprendí

Durante el desarrollo de este proyecto, comprendí cómo funcionan las bases de datos y por qué se diseñan de cierta manera. Descubrí que existe toda una metodología para evitar errores y lograr eficiencia mediante el diseño normalizado.

Dominando las Formas Normales. Lo más importante que aprendí fueron las Formas Normales, conceptos que previenen redundancias y anomalías. Con la Primera Forma Normal (1FN) descubrí que cada campo debe contener un solo valor atómico. Casi cometo el error de poner varios autores separados por comas, pero eso causaría anomalías de actualización al buscar o modificar información. La Segunda Forma Normal (2FN) me enseñó que todos los atributos deben depender completamente de la clave primaria. En mi tabla Prestamo usé un ID_Prestamo único en lugar de una clave compuesta (ID_Libro, ID_Estudiante), evitando así que datos como el nombre del estudiante se repitieran en cada préstamo. Con la Tercera Forma Normal (3FN) todo hizo "click": en lugar de duplicar nombre y nacionalidad del autor en cada libro (causando redundancia y anomalías de actualización), guardé solo el ID_Autor como clave foránea. Mi diagrama ER final reflejó estas relaciones claramente, mostrando las dependencias entre tablas.

En la práctica: implementación en C++. Traducir la teoría a código fue desafiante. Aprendí a usar struct para estructuras de datos y vector para colecciones en memoria. Las operaciones CRUD se convirtieron en funciones que construía diariamente. Diseñé un menú organizado con secciones como [AUTORES], [LIBROS], [PRÉSTAMOS], priorizando la experiencia del usuario. La validación fue crucial: implementé leerEntero() para IDs positivos, leerSoloDigitos() para ISBNs únicos, y validarFecha() con regex para formato YYYY-MM-DD. Un momento decisivo fue cuando el programa crasheó al leer un archivo corrupto: implementé try-catch en cargarDatos() para capturar excepciones durante el parsing, evitando que datos mal formados rompieran todo el sistema. Aprendí cin.clear() y cin.ignore() para limpiar el buffer tras entradas inválidas. La persistencia con fstream fue muy satisfactoria. Ver que mi programa guardaba datos usando el delimitador |, cerrarlo, reabrirlo y encontrar todo intacto fue emocionante. Crear la función split() para parsear líneas me enseñó sobre manipulación de cadenas a bajo nivel.

Reflexión y proyección futura. Este proyecto me transformó. Entendí que programar bien implica diseño cuidadoso desde el inicio. Diseñar una base de datos normalizada, validar datos robustamente y pensar en el usuario son habilidades que aplicaré en mi próximo proyecto de sistema de inventario. Lo más valioso fue entender el "por qué": ahora analizo cualquier aplicación preguntándome cómo está diseñada su base de datos. La programación es arte y ciencia: teoría sólida más creatividad e iteración constante hasta la perfección.