UNIVERSIDAD DEL VALLE DE GUATEMALA

Base de Datos 1

Sección 10

Catedrático: Erick Marroquín

"Proyecto 4"

Estuardo Castro – 23890

Roberto Nájera - 23781

Guatemala, 07 de junio de 2025

Reflexión

1. ¿Cuál fue el aporte técnico de cada miembro del equipo?

Roberto trabajó principalmente en la creación de la base de datos y las migraciones. También implementó los módulos CRUD usando Django y el ORM, lo cual permitió una interacción fluida con las tablas. Por otro lado, Estuardo se encargó de generar los datos de prueba que insertamos en la base (data.sql), además de diseñar y probar las vistas SQL que usamos en los índices. Ambos colaboramos en revisar y probar el funcionamiento del sistema completo.

2. ¿Qué decisiones estructurales se tomaron en el modelo de datos y por qué?

Desde el inicio sabíamos que necesitábamos un modelo robusto y bien conectado, así que diseñamos más de veinte tablas con relaciones uno a muchos y muchos a muchos. Usamos tablas intermedias para representar relaciones como la de libros con autores y libros con géneros. También separamos claramente a los usuarios, sus roles y permisos, para mantener el control de acceso bien definido.

3. ¿Qué criterios siguieron para aplicar la normalización?

Nos aseguramos de que cada tabla cumpliera con hasta tercera forma normal. Evitamos repetir datos, separamos atributos multivaluados y nos fijamos que no existieran dependencias transitivas. Por ejemplo, los géneros y autores se movieron a sus propias tablas y las relaciones fueron modeladas aparte para mantener todo limpio.

4. ¿Cómo estructuraron los tipos personalizados y para qué los usaron?

Creamos tres tipos personalizados en PostgreSQL. Uno para representar el estado físico del libro, otro para las calificaciones de los usuarios y otro compuesto para manejar precios con moneda. Los usamos porque nos daban más control desde la base de datos y facilitaban validaciones, además de que representaban mejor los datos que queríamos guardar.

5. ¿Qué beneficios encontraron al usar vistas para el índice?

Las vistas nos ayudaron a simplificar muchas consultas que de otra forma requerirían varios joins. Nos permitieron mostrar datos complejos en las pantallas de índice sin tener que complicar el código de Django. También fueron útiles para mantener una separación entre la lógica de negocio y la visualización de datos.

6. ¿Cómo se aseguraron de evitar duplicidad de datos?

Usamos restricciones de unicidad en los modelos y también en la base de datos. Algunos ejemplos son los correos electrónicos de los usuarios, los ISBN de los libros y las combinaciones de autor y libro en las relaciones. Además, validamos manualmente con Django ciertos casos para evitar que se repitieran registros sin sentido.

7. ¿Qué reglas de negocio implementaron como restricciones y por qué?

Implementamos varias restricciones importantes, como asegurarnos de que la fecha de devolución de un préstamo fuera después de la fecha del préstamo, o que la fecha de nacimiento de un usuario fuera anterior al día actual. También pusimos límites como que los precios fueran mayores a cero o que las páginas de un libro no fueran negativas. Estas reglas nos ayudaron a mantener la coherencia en los datos.

8. ¿Qué trigger resultó más útil en el sistema? Justifica.

El trigger más útil fue el que se encargaba de actualizar automáticamente la disponibilidad de una copia de libro cuando se hacía un préstamo o una devolución. Esto evitó tener que hacerlo manualmente en cada operación y ayudó a que el sistema se mantuviera sincronizado sin errores.

9. ¿Cuáles fueron las validaciones más complejas y cómo las resolvieron?

Las más complicadas fueron aquellas que dependían de otras fechas, como validar que una devolución ocurriera después del préstamo, o calcular cuándo aplicar una multa. Para resolverlo, combinamos validaciones en el modelo de Django con triggers en la base. De esa forma, nos aseguramos que las reglas se cumplieran tanto desde la app como desde la base directamente.

10. ¿Qué compromisos hicieron entre diseño ideal y rendimiento?

Hubo momentos en los que tuvimos que elegir entre un diseño perfectamente normalizado y consultas más simples. Por ejemplo, usar un array para los idiomas del libro fue una forma de simplificar sin tener que crear una tabla adicional. También aceptamos que ciertas vistas complejas podían tardar un poco más, pero preferimos mantener un modelo más limpio y lógico a largo plazo.