Análisis

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

Se eligió SQLAlchemy por su madurez y flexibilidad, ya que permite definir modelos con clases Python, manejar relaciones complejas como many-to-many de forma clara, y soportar tipos personalizados como ISBN. Además, se integra bien con FastAPI mediante Pydantic y permite usar distintas bases de datos como SQLite y PostgreSQL. Estos factores facilitaron una implementación ordenada y escalable sin complicaciones técnicas destacables.

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

La lógica master-detail se resolvió con relaciones bidireccionales entre libros y autores usando relationship() y una tabla intermedia. La API devuelve datos anidados para reflejar esta estructura, y una función específica maneja la asignación de autores a libros, permitiendo mantener la relación sincronizada y clara tanto al insertar como al consultar datos.

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

En la base de datos se definieron restricciones como formatos válidos para ISBN y email, unicidad de título e ISBN, y un mínimo de año de publicación. En el código se reforzaron estas reglas con validaciones en los modelos, incluyendo tipos personalizados, campos obligatorios y enumeraciones, asegurando integridad y coherencia desde ambas capas.

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

El uso de tipos personalizados permitió validar formatos como ISBN y email desde un solo lugar, lo que mejora la consistencia de los datos, facilita el mantenimiento, y aporta claridad al modelo. También refuerza la seguridad al trasladar reglas de validación directamente a la base de datos.

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

Usar una VIEW como base del índice permite encapsular la lógica compleja de consulta en una capa reutilizable y mantenible. También separa la lógica de presentación de la lógica de persistencia, facilitando su uso en distintas partes de la aplicación.

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

- Registros duplicados de libros o autores.
- Relaciones inválidas entre libros y autores (IDs inexistentes).
- Formatos incorrectos para ISBN o correo electrónico.

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

Aprendimos que mantener una separación clara permite que cada capa sea más fácil de probar, mantener y evolucionar. Esta división mejora la modularidad y la posibilidad de escalar o migrar cada capa por separado.

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

Este diseño escalaría bien gracias al uso de índices (automáticos en claves primarias y únicas), vistas optimizadas e integridad relacional.

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

Sí, porque el diseño es modular, con modelos bien definidos y separación de responsabilidades. En un entorno de microservicios, se podrían separar autores, libros y autenticación en servicios independientes que se comuniquen vía APIs.

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

La vista se puede consultar directamente desde la API o desde scripts para generación de reportes. También podría exponerse como un endpoint especializado (por ejemplo, /libros/detalles) que devuelva datos ya combinados de varias tablas, optimizando el rendimiento y reduciendo la lógica del cliente.

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

- Usar una tabla intermedia para representar la relación many-to-many entre libros y autores.
- Definir dominios y enums para mejorar la validación de datos en la base.
- Usar claves primarias auto-incrementales (SERIAL) para facilitar referencias.
- Nombrar tablas y columnas de forma clara y consistente.

Estas decisiones favorecen la integridad referencial, la claridad y la extensibilidad del modelo.

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

Archivo README.md y diagrama entidad relación explicando las relaciones entre tablas y ejemplos de uso.

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

- Definimos una clave primaria compuesta (libro_id, autor_id) en autores_libros, lo que impide duplicaciones.
- Usamos validaciones en la API para evitar crear relaciones repetidas.
- Aprovechamos restricciones de clave foránea con ON DELETE CASCADE para mantener la consistencia si se elimina un autor o libro.