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

- **José Sánchez:** Diseñó el modelo lógico y físico, definió reglas de negocio, tipos personalizados y el diagrama ER.
- **Eliazar Canastuj:** Se encargó de la creación de tablas en SQL, basándose en el modelo diseñado.
- **Andre Pivaral:** Desarrolló la lógica para insertar automáticamente más de 1000 registros de prueba de manera coherente.
- **Ángel Mérida:** Implementó vistas SQL y triggers funcionales para el sistema.
- **Pablo Cabrera:** Diseñó las interfaces frontend para los CRUDs y la visualización de datos.
- **Luis Palacios:** Desarrolló el backend con Flask para los CRUDs y reportes, incluyendo filtrado y exportación en varios formatos.

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

Se optó por un modelo altamente relacional, con entidades separadas para usuarios, canciones, álbumes, géneros, colaboraciones, etc., para favorecer la normalización y permitir relaciones complejas como colaboraciones múltiples y métricas por canción.

3. ¿Qué criterios siguieron para aplicar la normalización?

Se aplicaron las tres primeras formas normales:

- 1FN: cada columna contiene un valor atómico.
- 2FN: se eliminaron dependencias parciales.
- 3FN: se eliminaron dependencias transitivas (por ejemplo, los géneros se normalizaron en una tabla aparte).

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

Se crearon tipos personalizados como clave_musical, formato_archivo y rol_usuario para mantener la integridad semántica y facilitar validaciones automáticas desde PostgreSQL.

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

Permitieron simplificar el frontend, encapsular joins complejos y evitar exponer directamente tablas sensibles, facilitando la implementación de filtros y reportes.

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

Usando:

- Llaves primarias y únicas en campos como email o nombre de género.
- Validaciones desde el ORM y SQL.
- Vistas que consolidan datos sin permitir edición directa.

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

- CHECK en puntajes (1 a 5).
- Llaves foráneas para asegurar relaciones válidas entre álbumes, canciones y usuarios.
- Tipos personalizados restringen valores posibles (e.g., clave_musical).

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

Un trigger que actualiza automáticamente las métricas de una canción (reproducciones/likes) resultó clave para mantener datos sincronizados sin intervención manual.

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

Validar la combinación de artista y álbum al crear canciones, especialmente en el frontend con selects dependientes. Se resolvió sincronizando las opciones mediante JS y mapas de relaciones desde el backend.

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

Se evitó usar demasiadas subconsultas en vistas para favorecer el rendimiento, y se aceptó cierta duplicación lógica en vistas para facilitar el filtrado en reportes.

11. ¿Qué estrategia usaron para distribuir los datos de prueba?

Se agruparon por bloques: álbumes 1–10 para Taylor Swift, 11–20 para Bad Bunny, etc., y se automatizó la inserción para distribuir géneros, duraciones y métricas realistas.

12. ¿Qué tablas fueron más difíciles de poblar y por qué?

Las relacionadas con métricas, historial y colaboraciones, porque requerían validaciones previas, fechas consistentes y referencias cruzadas válidas.

13. ¿Qué harían diferente si pudieran rediseñar la base?

Agregar más modularidad en colaboraciones y métricas, y permitir múltiples licencias por canción para representar mejor la industria real.

14. ¿Cómo estructuraron los 3 CRUDs para mantener la consistencia?

Cada CRUD está dividido en:

- Vista SQL como índice.
- Operaciones de inserción y edición desde tabla base.
- Formularios y controles validados con JS y backend para mantener relaciones coherentes.

15. ¿Qué aprendizaje obtuvieron del uso del ORM?

Permite mantener coherencia entre modelo lógico y código, simplifica la escritura de queries complejas y agiliza el desarrollo, aunque requiere comprender bien las relaciones y serialización.

16. ¿Cómo reutilizarían su diseño en otros proyectos?

La arquitectura modular con vistas, tipos personalizados y ORM es fácilmente adaptable a otros sistemas como bibliotecas musicales, plataformas educativas, etc.

17. ¿Qué tan escalable consideran que es el sistema?

Muy escalable: permite añadir más métricas, tipos de usuario, funcionalidades (como reseñas por playlist), sin romper el modelo base.

18. ¿Qué limitaciones encontraron al usar vistas?

No permiten edición directa en algunos casos, por lo que las operaciones de edición/inserción deben redirigirse a las tablas base, agregando complejidad.

19. ¿Qué estrategias implementaron para controlar integridad referencial?

Uso riguroso de llaves foráneas, restricciones ON DELETE CASCADE donde aplicaba y validaciones en el backend para prevenir inserts inconsistentes.

20. ¿Qué impacto tuvo la coevaluación en su percepción del trabajo en equipo?

Permite reconocer el esfuerzo de cada miembro, motiva a contribuir equitativamente y mejora la comunicación y coordinación dentro del grupo.