1. En el método Crear de la clase JugadorService, ¿por qué se utiliza SCOPE_IDENTITY() en la consulta SQL y qué beneficio aporta al código?
   Se usa para obtener el ID del nuevo jugador recién insertado en la base de datos. Esto es útil para seguir trabajando con ese jugador sin tener que volver a hacer una consulta.

2. En el método Eliminar del servicio de jugadores, ¿por qué se verifica la existencia de
   elementos en el inventario antes de eliminar un jugador y qué problema está
   previniendo esta comprobación?
   Se verifica para ver que no se eliminen jugadores que aun tienen cosas en su inventario, para que no existan errores de credenciales.

3. ¿Qué ventaja ofrece la línea using var connection = _dbManager.GetConnection();
   frente a crear y cerrar la conexión manualmente? Menciona un posible problema que
   podría ocurrir si no se usara esta estructura.
   Asegura que la conexión se cierre automáticamente, incluso si ocurre un error. Sin esta estructura, podría dejar conexiones abiertas, causando fugas de recursos.

4. En la clase DatabaseManager, ¿por qué la variable _connectionString está marcada
   como readonly y qué implicaciones tendría para la seguridad si no tuviera este
   modificador?
   Impide que se modifique después de ser asignada. Esto mejora la seguridad al evitar que el string de conexión sea alterado en tiempo de ejecución.

5. Si quisieras agregar un sistema de logros para los jugadores, ¿qué cambios
   realizarías en el modelo de datos actual y qué nuevos métodos deberías implementar en los servicios existentes?
   Agregaria una nueva tabla `Logros`. En el servicio, agregaria métodos como `AgregarLogro`, `ObtenerLogrosPorJugador`

6. ¿Qué sucede con la conexión a la base de datos cuando ocurre una excepción
dentro de un bloque using como el que se utiliza en los métodos del JugadorService?
La conexión se cierra automáticamente, incluso si ocurre una excepción.

7. En el método ObtenerTodos() del JugadorService, ¿qué ocurre si la consulta SQL no
devuelve ningún jugador? ¿Devuelve null o una lista vacía? ¿Por qué crees que se
diseñó de esta manera?
Devuelve una lista vacía, no `null`. Esto evita errores de tipo `NullReferenceException` al recorrer los resultados, y hace que el código sea más seguro.

8. Si necesitaras implementar una funcionalidad para registrar el tiempo jugado por
cada jugador, ¿qué cambios harías en la clase Jugador y cómo modificarías los
métodos del servicio para mantener actualizada esta información?
Añadirias una propiedad como `TiempoJugado` en la clase `Jugador`. En el servicio, agregaría una lógica para incrementar ese valor al finalizar sesiones de juego.

9. En el método TestConnection() de la clase DatabaseManager, ¿qué propósito
cumple el bloque try-catch y por qué es importante devolver un valor booleano en
lugar de simplemente lanzar la excepción?
El try-catch evita que la aplicación se detenga por una excepción. Retornar un bool permite al código saber si la conexión es válida sin tener que lanzar errores.

10. Si observas el patrón de diseño utilizado en este proyecto, ¿por qué crees que se
separaron las clases en carpetas como Models, Services y Utils? ¿Qué ventajas
ofrece esta estructura para el mantenimiento y evolución del proyecto?
Separar en `Models`, `Services` y `Utils` aplica el principio de responsabilidad única, facilitando el mantenimiento, pruebas y evolución del proyecto.

11. En la clase InventarioService, cuando se llama el método AgregarItem, ¿por qué es
necesario usar una transacción SQL? ¿Qué problemas podría causar si no se
implementara una transacción en este caso?
La transacción asegura que se agreguen todos los ítems o ninguno. Si falla a mitad, evita un estado inconsistente. Sin transacción, se podria tener inventarios a medias.

12. Observa el constructor de JugadorService: ¿Por qué recibe un DatabaseManager
como parámetro en lugar de crearlo internamente? ¿Qué patrón de diseño se está
aplicando y qué ventajas proporciona?
Se aplica Inversión de Dependencias. Permite inyectar diferentes implementaciones de DataBaseManager

13. En el método ObtenerPorId de JugadorService, ¿qué ocurre cuando se busca un ID
que no existe en la base de datos? ¿Cuál podría ser una forma alternativa de
manejar esta situación?
Si el ID no existe, se retorna null. Como alternativa se puede lanzar una excepción o retornar un objeto con un mensaje de "no encontrado".

14. Si necesitas implementar un sistema de "amigos" donde los jugadores puedan
conectarse entre sí, ¿cómo modificarías el modelo de datos y qué nuevos métodos
agregarías a los servicios existentes?
Crearia una tabla intermedia Amistades `(JugadorId, AmigoId)` para relaciones muchos a muchos. Agregaria métodos como `AgregarAmigo`, `EliminarAmigo`, `ListarAmigos`.

15. En la implementación actual del proyecto, ¿cómo se maneja la fecha de creación de
un jugador? ¿Se establece desde el código o se delega esta responsabilidad a la
base de datos? ¿Cuáles son las ventajas del enfoque utilizado?

La fecha de creación del jugador se asigna desde el código. Usar la base de datos sería mejor porque garantiza que todos los registros tengan una fecha uniforme y segura.

16. ¿Por qué en el método GetConnection() de DatabaseManager se crea una nueva
instancia de SqlConnection cada vez en lugar de reutilizar una conexión existente?
¿Qué implicaciones tendría para el rendimiento y la concurrencia?

Cada vez se crea una nueva conexión para evitar problemas de concurrencia. Reutilizar conexiones podría causar errores si varias operaciones ocurren al mismo tiempo.

17. Cuando se actualiza un recurso en el inventario, ¿qué ocurriría si dos usuarios
intentan modificar el mismo recurso simultáneamente? ¿Cómo podrías mejorar el
código para manejar este escenario?

Podría haber conflictos si dos usuarios modifican al mismo tiempo. Una mejora sería usar control de concurrencia optimista

18. En el método Actualizar de JugadorService, ¿por qué es importante verificar el valor
de rowsAffected después de ejecutar la consulta? ¿Qué información adicional proporciona al usuario?

Indica si realmente se actualizó algo. Si es 0, probablemente el ID no existe o no hubo cambios. Esto ayuda a informar mejor al usuario.

19. Si quisieras implementar un sistema de registro (logging) para seguir todas las
operaciones realizadas en la base de datos, ¿dónde colocarías este código y cómo
lo implementarías para afectar mínimamente la estructura actual?

Se puede crear una clase `Logger` en `Utils` e inyectarla en los servicios. El código de logging se puede poner justo antes y después de operaciones críticas, y escribir en un archivo o tabla.

20. Observa cómo se maneja la relación entre jugadores e inventario en el proyecto. Si
necesitaras agregar una nueva entidad "Mundo" donde cada jugador puede existir
en múltiples mundos, ¿cómo modificarías el esquema de la base de datos y la estructura del código para implementar esta funcionalidad?

Crearia la tabla `Mundos` y una tabla intermedia `JugadorMundo (JugadorId, MundoId)` luego adaptaria los métodos para manejar la asignación y recuperación de mundos por jugador.

21. ¿Qué es un `SqlConnection` y cómo se usa?

Es una clase que permite conectarte a una base de datos SQL Server. Se usa para abrir, ejecutar comandos y cerrar conexiones con la base de datos.

22. ¿Para qué sirven los `SqlParameter`?

Sirve para pasar valores a comandos SQL de forma segura, evitando inyecciones SQL y asegurando el tipo de dato correcto.