CUESTIONARIO:

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 SCOPE_IDENTITY() para obtener el ID del jugador recién insertado. Permite trabajar con ese ID sin riesgo de confusión en operaciones simultáneas.

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?

Para evitar eliminar un jugador con inventario activo. Se previene un error de integridad referencial en la base de datos.

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. Si no se usa, podrías dejar conexiones abiertas y saturar el servidor SQL.

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?

readonly evita que la cadena se modifique accidentalmente, protegiendo la seguridad y consistencia de la conexión.

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?

Agregar una tabla Logros y métodos como AgregarLogro, ObtenerLogrosPorJugador, para gestionar los logros de cada jugador.

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 aunque haya un error, evitando fugas de recursos.

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, para evitar errores al usar la lista y no tener que comprobar si es null.

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?

Agregar una propiedad TiempoJugado y actualizarla en los métodos que registran actividad del jugador.

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?

try-catch evita que la app se caiga y permite informar si la conexión fue exitosa o fallida de forma controlada.

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?**

Organiza el código por responsabilidad. Facilita su mantenimiento y evolución sin confusión ni duplicación de código.

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 todo el proceso de agregar un ítem sea completo y coherente. Sin ella, los datos quedarían incompletos si hay errores.

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 inyección de dependencias. Esto mejora la reutilización del código y facilita las pruebas.

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?**

Devuelve null. Otra opción sería lanzar una excepción o devolver un objeto con un mensaje de error.

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?**

Crear una tabla Amigos con JugadorId y AmigoId. Métodos: 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?**

Se delega a la base de datos con GETDATE(). Esto asegura coherencia y evita errores del reloj del sistema local.

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?**

Evita conflictos por reutilización en entornos multicliente. Mejor manejo de la concurrencia y estabilidad.

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?**

Podrían sobrescribirse los cambios. Se puede usar control de versiones con rowversion o timestamps.

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?**

Confirma si el jugador fue realmente actualizado o si no se encontró el ID en la base de datos.

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?

Crear clase Logger y llamarla desde los métodos de servicio. Guardar en archivo o tabla Logs sin alterar la lógica existente.

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?

Crear tablas Mundo y JugadorMundo. En el código, agregar clases para Mundo y métodos para vincular jugadores a mundos.

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

Es un objeto que representa una conexión a SQL Server. Se usa para abrir, usar y cerrar conexión a la base de datos.

22. ¿Para qué sirven los SqlParameter?

Se usan para enviar datos seguros a las consultas SQL, evitando errores y ataques de inyección.