Cuestionario sobre proyecto parcial

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?**
   SCOPE_IDENTITY() se usa para obtener el ID generado automáticamente al insertar un nuevo jugador. Asegura que se obtenga el ID correcto en ambientes con múltiples inserciones 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?**
   Se verifica la existencia de elementos del inventario para evitar errores de integridad referencial y eliminar datos relacionados. Previene inconsistencias y errores lógicos.

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.**
   Usar using asegura que la conexión se cierre automáticamente. Sin esto, podrían quedar conexiones abiertas causando saturación y fallos en la base de datos.

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?**
   Está marcada como readonly para evitar que se modifique accidentalmente. Mejora la seguridad del acceso a la base de datos y mantiene la configuración inmutable.

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?**
   Se requeriría una tabla Logros relacionada con Jugadores. Se deben crear métodos como AgregarLogro, ObtenerLogrosPorJugador, etc.

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 libera automáticamente aunque ocurra una excepción, evitando pérdidas de recursos o conexiones colgadas.
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 esa forma?**
   Devuelve una lista vacía. No lanza excepción porque no es un error, solo no hay datos disponibles. Facilita el manejo lógico sin errores innecesarios.

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?**
   Agregarías una propiedad TiempoJugado en la clase Jugador y métodos como RegistrarTiempoJugado en el servicio.

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?**
   Permite capturar errores sin detener la aplicación y devolver true o false según el resultado. Es útil para validaciones sin romper el flujo.

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?**
    Mejora la organización, facilita el mantenimiento y promueve la separación de responsabilidades, haciendo el proyecto escalable y entendible.

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?**
    Una transacción asegura que todos los pasos se completen correctamente o ninguno. Sin transacción, podrían quedar datos incompletos o inconsistentes.

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 (DI). Facilita pruebas, reutilización y separación de responsabilidades.

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 o lanza una excepción dependiendo de la implementación. Alternativa: retornar un resultado controlado para evitar errores en el cliente.

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?**
    Agregar una tabla Amigos que relacione jugadores. 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?**
    Idealmente se delega a la base con GETDATE(). Ventajas: mayor precisión, consistencia y evita errores en cliente.

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 en uso concurrente. Reutilizar conexiones puede causar errores si se usan al mismo tiempo desde diferentes hilos.

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 o pérdida de datos. Soluciones: usar bloqueo optimista con marcas de tiempo o control de concurrencia.

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 la operación afectó alguna fila. Si el valor es 0, probablemente no existía el jugador; ayuda en la validación del resultado.

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?**
    Usaría una clase Logger común o middleware que registre acciones en tabla de logs o archivo. Llamado desde los servicios sin afectar su lógica.

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 tabla Mundo y tabla intermedia JugadorMundo para generar la relación y modificar servicios para soportar múltiples mundos por jugador.

21. **¿Qué es un SqlConnection y cómo se usa?**
    Clase para establecer una conexión con SQL Server. Se abre, se ejecutan comandos, y luego se cierra. Se recomienda usar con using.

22. **¿Para qué sirven los SqlParameter?**
    Para enviar parámetros de forma segura a consultas SQL. Previenen inyección SQL y problemas de formato en los valores enviados.