Cuestionario a responder:

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 recién generado** tras insertar un nuevo jugador. Esto permite asignarlo al objeto en memoria sin hacer una segunda 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?

Para evitar errores de integridad referencial. Si se borra un jugador con inventario, se rompería la relación. Previene excepciones y pérdida de datos relacionados.

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.

Cierra y libera automáticamente la conexión incluso si hay errores. Sin using, podrías dejar conexiones abiertas, lo cual causa bloqueos o consumo excesivo 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?

Evita que sea modificada accidentalmente. Si no fuera readonly, alguien podría cambiarla en tiempo de ejecución, comprometiendo seguridad y estabilidad.

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?

- Crear tabla Logros y JugadorLogro (relación N:M).
- Agregar modelo Logro.
- Métodos en el servicio: AgregarLogroAJugador, ObtenerLogrosDelJugador, 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?

Se cierra igual. using garantiza que el recurso se libere pase lo que pase, incluso si hay un throw.

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?

Una **lista vacía**, no null. Esto evita errores como NullReferenceException y facilita el manejo en la interfaz.

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?

- Agrega una columna TiempoJugado en Jugador.
- Actualiza modelo y métodos ActualizarTiempo.
- Mostrar tiempo en la interfaz o gráficos.

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?

Evita que la app crashee si hay error de conexión. Devolver un booleano permite manejar el error suavemente desde la interfaz.

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 (patrón MVC). Mejora el mantenimiento, escalabilidad y comprensión 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?

Para asegurar que todas las operaciones ocurran juntas. Sin transacción, podrías insertar parcialmente datos y dejar el sistema en estado inconsistente.

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?

Aplica **inyección de dependencias**, un patrón que mejora la testabilidad, 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. Se puede manejar mostrando un mensaje o lanzando una excepción controlada.

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 tabla Amistades con JugadorId1, JugadorId2.
- Métodos en el servicio: 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 desde la base de datos con un DEFAULT GETDATE(). Esto asegura que siempre tenga un valor correcto sin depender del código.

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?

Idealmente desde la base de datos con un DEFAULT GETDATE(). Esto asegura que siempre tenga un valor correcto sin depender del código.

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?

Pueden sobrescribir cambios. Se puede resolver con **control de concurrencia optimista**, agregando una columna RowVersion o timestamp.

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?

Permite saber si realmente se modificó un registro. Si 0, puede significar que no existía o no hubo cambios.

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?

Usar una clase Logger y registrar en archivo o base de datos. Llamar desde servicios sin alterar su estructura, usando eventos o decoradores.

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 relación JugadorMundo.
- Modelo Mundo, métodos como AgregarJugadorAMundo, ListarMundosPorJugador.

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

Es la clase que permite conectarse a una base de datos SQL Server. Se usa con using para ejecutar comandos, por ejemplo: Es la clase que permite conectarse a una base de datos SQL Server. Se usa con using para ejecutar comandos, por ejemplo:

```
using var connection = new SqlConnection(connectionString);

connection.Open();
```

22. ¿Para qué sirven los SqlParameter?

Previenen inyecciones SQL y permiten pasar valores de forma segura a los comandos SQL:

```
command.Parameters.AddWithValue("@id", jugador.Id);
```