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?**

SCOPE_IDENTITY() se utiliza para obtener el último valor de una columna de identidad generada automáticamente en el mismo ámbito de ejecución de la consulta SQL. Esto es útil para recuperar el Id del jugador recién creado en la base de datos, asegurando que el valor corresponde exactamente a la inserción realizada. El beneficio principal es evitar conflictos en escenarios repetitivas, donde múltiples acciones podrían ocurrir simultáneamente.

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 inconsistencias en la base de datos. Si se eliminara un jugador podrían quedar registros huérfanos en la tabla de inventario, lo que violaría la integridad referencial. Previene la falta de coherencia en el código.

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.

La estructura using garantiza que la conexión se cierre y libere automáticamente al salir del bloque. Un posible problema sería olvidar cerrar la conexión manualmente, lo que podría llevar a un agotamiento de conexiones en el pool de 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?

Garantiza que su valor solo pueda asignarse en el constructor o en su declaración inicial. Una implicación seria que alguien externo pueda modificar.

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?

Crearia una nueva tabla de los Logros con columnas como Id, Nombre y FechaCreacion. También crearía una tabla intermedia para relacionar jugadores con sus logros.

En los servicios crear nuevo logro, elimar.

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?

El objeto gestionado se libera automáticamente al salir del bloque. Esto asegura que la conexión se cierre correctamente.

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?

Si la consulta SQL no devuelve ningún jugador, el método devuelve una lista vacía en lugar de null. Se pudo diseñar de esta manera para la estabilidad del código.

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?

Pues agregarle una propiedad a la clase, como: tiempoJugado.

Crear un método que actualice el tiempo en la base de datos, desde el momento que entra a la aplicación del juego hasta que sale, incluso se podría agregar si entra durante un rato en el día y sumarle si entra otro tiempo en el juego y darle el total del tiempo.

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?

Se utiliza para cualquier excepción que pueda suceder a la hora de la conexión. Es importante ya que el método se vuelve más reutilizable y sencillo.

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?

Se separan para un mejor entendimiento y un código más organizado para trabajar.

Saber en donde se encuentra cada clase facilita cualquier evolución en el 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?

Son necesarias porque hacen que los datos en la base de datos cuenten con seguridad y consistencia. Sin una transacción podrían aparecer datos 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?

Lo recibe para aplicar el patrón de diseño Dependency injection, facilitando la prueba unitaria y mejorando la flexibilidad.

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, el método devuelve null. Esto permite al llamador manejar la ausencia del jugador de manera explícita. Una alternativa sería lanzar una excepción personalizada, como JugadorNoEncontradoException, para que el usuario se dé cuenta de una vez cual es el 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?

Pues crear una tabla llamada amigos donde vayan datos como el id, nombre, etc.

Agregaría Agregar, Eliminar, Listar, etc.

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 maneja automáticamente.

Se delega a la base de datos, ya que no se especifica en el código al insertar un jugador.

Que la fecha sea precisa y consistente.

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?

Para garantizar que cada operación tenga su propia conexión independiente.

Puede llevar a un consumo excesivo de recursos del servidor de base de datos.

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?

Si dos usuarios intentan modificar el mismo recurso simultáneamente, podría ocurrir una condición de carrera, resultando en datos inconsistentes.

Implementar bloqueos optimistas o pesimistas.

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?

Verificar rowsAffected asegura que la operación afectó al menos un registro. Esto informa al usuario si el jugador existía o no.

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?

Agregar el registro en DatabaseManager para capturar todas las operaciones de base de datos.

Usar una biblioteca como Serilog o NLog para registrar consultas, parámetros y resultados.

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 una tabla Mundos con columnas. Crear una tabla intermedia JugadorMundos con columnas JugadorId y MundoId para representar la relación muchos a muchos.

Agregar métodos en JugadorService para asociar jugadores con mundos (AgregarMundo, ObtenerMundos). Crear un nuevo servicio MundoService para gestionar los mundos.

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

Es una clase que permite la conexión a una base de datos en Sql.

Se usa para trabajar con las conexiones.

22. ¿Para qué sirven los SqlParameter?

se usa para pasar parámetros a consultas SQL de manera segura.