1. ¿Por qué se usa SCOPE_IDENTITY() en Crear de JugadorService?
Se emplea SCOPE_IDENTITY() para recuperar el ID del jugador recién creado en la misma transacción. Esto asegura que obtengamos el valor correcto, especialmente cuando varias inserciones suceden al mismo tiempo, y evita asignaciones incorrectas en el sistema.

2. ¿Por qué se revisa el inventario antes de eliminar un jugador?
Se comprueba que el jugador no tenga objetos en su inventario para evitar eliminar datos aún en uso. Así se protegen la integridad y coherencia de la base de datos, previniendo errores y registros huérfanos.

3. ¿Ventaja de usar using var connection?
El using cierra la conexión automáticamente cuando termina su uso, incluso si ocurre un error. Si no se utilizara esta estructura, podríamos olvidar cerrar la conexión, generando saturación de recursos y fallos en el sistema.

4. ¿Importancia de readonly en _connectionString?
Marcar la cadena de conexión como readonly garantiza que no se pueda cambiar después de asignarla, protegiendo la seguridad del proyecto. Si se pudiera modificar, existiría el riesgo de redireccionar datos hacia destinos no autorizados.

5. ¿Cómo agregar un sistema de logros para jugadores?
Habría que crear nuevas tablas de Logros y de relaciones JugadorLogros. Además, se añadirían métodos en los servicios para asignar logros a jugadores y permitir consultarlos.

6. ¿Qué pasa con la conexión si ocurre una excepción dentro de using?
Aunque ocurra un error, la conexión a la base de datos se cerrará correctamente gracias al bloque using. Esto garantiza que no queden conexiones abiertas consumiendo recursos.

7. ¿Qué pasa si no hay jugadores en ObtenerTodos()?
El método retorna una lista vacía, no null. Esto facilita el uso del resultado, ya que evita tener que comprobar si el objeto existe antes de trabajar con él.

8. ¿Cómo registrar el tiempo jugado por cada jugador?
Se debería añadir un nuevo campo como TiempoJugado en la clase Jugador y en la base de datos. También habría que modificar los métodos para actualizar y consultar esta información cuando el jugador esté activo.

9. ¿Qué función cumple el try-catch en TestConnection()?
El bloque try-catch permite capturar errores de conexión y devolver simplemente un false, sin que el programa se detenga. Esto da más control sobre la respuesta ante un fallo.

10. ¿Por qué separar el código en Models, Services y Utils?
Dividir el código en carpetas ayuda a organizarlo mejor, facilita su mantenimiento y hace más sencillo agregar nuevas funcionalidades en el futuro, manteniendo todo claro y modular.

11. ¿Por qué usar una transacción en AgregarItem?
La transacción asegura que todas las operaciones relacionadas se completen correctamente o no se haga ninguna. Sin transacciones, podría haber registros incompletos o inconsistencias si ocurre un error a mitad del proceso.

12. ¿Por qué JugadorService recibe DatabaseManager como parámetro?
Esto es un ejemplo de inyección de dependencias. Ayuda a que el servicio no dependa de crear sus propias conexiones, haciendo el código más flexible y más fácil de probar.

13. ¿Qué pasa si se busca un ID inexistente en ObtenerPorId?
El método devolverá null. Una opción alternativa podría ser lanzar una excepción

específica o retornar un objeto vacío, dependiendo de cómo se quiera gestionar el error.

14. ¿Cómo agregar un sistema de "amigos" entre jugadores?
Se necesitaría una nueva tabla Amigos que relacione jugadores entre sí. También se agregarían métodos para agregar, eliminar y listar amigos dentro del JugadorService.

15. ¿Cómo se maneja la fecha de creación del jugador?
No parece gestionarse directamente en el proyecto. Si se controlara desde la base de datos, se garantizaría mayor consistencia; desde el código, se tendría un control más personalizado de los datos.

16. ¿Por qué crear una nueva conexión en GetConnection() cada vez?
Generar una nueva conexión evita compartir objetos entre procesos o usuarios, lo cual podría provocar errores de concurrencia. Reutilizar conexiones sería arriesgado en entornos multitarea.

17. ¿Qué pasa si dos usuarios modifican un recurso al mismo tiempo?
Puede producirse un conflicto y sobrescribir cambios. Para evitarlo, se puede implementar control de concurrencia, como usar marcas de tiempo o versiones de los registros.

18. ¿Por qué verificar rowsAffected después de actualizar?
Porque así sabemos si realmente se hizo algún cambio. Esto permite informar al usuario si la actualización fue exitosa o si, por ejemplo, el ID del jugador no existía.

19. ¿Cómo implementar un sistema de logging?
Podríamos agregar registros en cada operación importante (insertar, actualizar, eliminar), usando una clase especial que guarde los eventos en un archivo de texto o una base de datos de logs.

20. ¿Cómo agregar un sistema de mundos para jugadores?
Se debería crear una tabla Mundo y una tabla intermedia para relacionar jugadores y mundos. También habría que adaptar los modelos y servicios para gestionar esta nueva relación.

21. ¿Qué es un SqlConnection?
Es un objeto que permite abrir una comunicación directa entre la aplicación y un servidor de base de datos SQL Server para ejecutar consultas o actualizaciones.

22. ¿Para qué sirven los SqlParameter?
Sirven para enviar valores seguros a las consultas SQL. Ayudan a evitar inyecciones de código malicioso y facilitan el manejo correcto de tipos de datos en las operaciones.