1. **Uso de SCOPE_IDENTITY() en Crear**
   Se usa para obtener el ID que la base de datos genera automáticamente al insertar un nuevo jugador. Así mantenemos sincronizado el objeto en memoria con el registro real en la base de datos.

2. **Verificación de inventario antes de eliminar**
   Antes de borrar un jugador, revisamos que no tenga ítems en su inventario. Esto evita errores y protege la integridad de la base de datos.

3. **Ventaja de using var connection**
   El using asegura que la conexión se cierre automáticamente, incluso si ocurre un error. Así evitamos fugas de recursos.

4. **Variable _connectionString como readonly**
   Hacerla readonly garantiza que no pueda cambiarse después de ser inicializada, protegiendo la conexión de cambios accidentales o maliciosos.

5. **Sistema de logros para jugadores**
   Habría que crear una nueva tabla de "Logros" y posiblemente una tabla intermedia para asociarlos a los jugadores, junto con métodos para agregar y consultar logros.

6. **Excepción en bloque using**
   Aunque haya una excepción, el using se encarga de cerrar la conexión porque SqlConnection implementa IDisposable.

7. **ObtenerTodos() sin resultados**
   Cuando no hay jugadores, se devuelve una lista vacía, no null. Esto es más seguro y evita errores por referencias nulas.

8. **Registrar tiempo jugado**
   Se añadiría una propiedad TiempoJugado al jugador y métodos para actualizarlo. También podríamos usar un temporizador para registrar sesiones.

9. **TestConnection() con try-catch**
   El try-catch atrapa cualquier error al intentar conectar. Así el programa no se cae y podemos manejar el error de forma controlada.

10. **Separación en carpetas**
    Organizar por carpetas (Models, Services, Utils) hace el proyecto más claro, fácil de mantener y con menos dependencias entre componentes.

11. **Transacción en AgregarItem**
    Se usa una transacción para asegurar que todas las operaciones se completen juntas. Si algo falla, se deshace todo para evitar inconsistencias.

12. **DatabaseManager como parámetro**
    Inyectarlo como parámetro reduce el acoplamiento y hace el sistema más flexible y fácil de probar (por ejemplo, usando mocks).

13. **ObtenerPorId con ID inexistente**
    Cuando no encuentra un jugador, devuelve null. Otra opción podría ser lanzar una excepción específica o usar un patrón como Maybe.

14. **Sistema de "amigos"**
    Sería necesario crear una tabla "Amistades" y agregar métodos para añadir y listar amigos de un jugador.

15. **Fecha de creación del jugador**
    Lo ideal es que la base de datos establezca automáticamente la fecha usando GETDATE(), asegurando coherencia y evitando problemas de zonas horarias.

16. **Nueva instancia de SqlConnection**
    Se crea una nueva conexión cada vez para asegurar independencia entre hilos. Además, el "connection pooling" ya gestiona las conexiones abiertas de forma eficiente.

17. **Actualización simultánea**
    Puede haber conflictos si varios usuarios actualizan al mismo tiempo. Se solucionaría usando transacciones o control de concurrencia optimista.

18. **Verificación de rowsAffected**
    Revisar rowsAffected nos dice si la operación tuvo efecto o si, por ejemplo, no existía el registro que queríamos actualizar.

19. **Sistema de logging**
    Se puede implementar con un servicio ILogger para registrar operaciones importantes, errores y flujos del sistema.

20. **Agregar entidad "Mundo"**
    Habría que crear una tabla "Mundos" y otra de relación "JugadorMundo", además de sus modelos y métodos en los servicios.

21. **SqlConnection**
    Es el objeto que usamos para abrir una conexión con SQL Server y ejecutar consultas o transacciones.

22. **SqlParameter**
    Se usa para construir consultas seguras (previniendo inyección SQL) y especificar correctamente los tipos de datos enviados a la base.