Preguntas del parcial 2 de Programación

EDUARDO GABRIEL VISONI MORALES

0905-22-1146

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?

Para obtener el valor del identificador único (**Id**) generado automáticamente por la base de datos al insertar un nuevo registro en la tabla **Jugadores**, y el resultado no se ve afectado por el hecho de que se ingresen otros datos en otras tablas. Su ventaja es que es muy exacto en la obtención precisa del registro recién insertado y asegura que el código sea preciso y seguro

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?

Verifica la integridad referencial (tablas relacionadas entre sí por una FOREIGN KEY, esto es, que, si una tabla está relacionada con otra, esa referencia debe siempre apuntar a un dato que exista), para que los datos del inventario no queden "Huérfanos" y que haya incoherencia en la lógica y se pierda el control de esos datos (¿Se deben eliminar? ¿Transferir?).

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 ventaja apunta a Using, ya que asegura cerrar la conexión de la base de datos automáticamente al finalizar su uso, sin tener que escribir conection.close(); al igual maneja excepciones y deja el código más limpio y se manejan varios problemas, como por ejemplo el dejar abierta la conexión, que por múltiples peticiones simultaneas, satura las conexiones disponibles y puede dejar la aplicación "colgada".

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?

Se marca como readonly para proteger la información en la inicialización de la base de datos en conexión con C# y que la cadena de conexión no cambie a partir de ese

momento, ya que si no fuera readonly podría haber múltiples problemas, como cambiar el destino de conexión a otra base de datos (puede ser maliciosa) y comprometer datos privados, etc.

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 agregarían las tablas de logros y logros de los jugadores, también nuevas clases como Logro y JugadorLogro, y dentro de esas clases nuevos métodos, como crear y listar logros, o asignar logros y obtener los logros de un jugador.

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?

Cuando ocurra, el Using hará que se cierre automática y correctamente la conexión con la base de datos y garantiza que no queden recursos abiertos o vulnerables, y alteración no deseada

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 no devuelve ningún valor, muestra una lista vacía, por múltiples razones como por ejemplo, es mas intuitivo para los desarrolladores, evita errores de referencia nula, etc. Se hizo de esa manera ya que es mas simple el manejo del resultado y evita algunos errores.

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?

En la clase jugador se agregaría la propiedad tiempojugado, tipo Timespan o int en minutos y se agregaría en la base de datos una columna donde se guarde esa informacion, y luego se agregarían métodos para actualizar esos datos cada vez que un jugador entra, y sale de la sesión.

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?

Lo que hace es capturar los errores de la conexión de la base de datos. Es importante el valor booleano es bueno en lugar de una excepción, ya que es mas amigable con el usuario y hace que el código sea más fácil de mantener, ya que simplemente devolvería un True o False, es mucho más fácil tanto para el código como para el usuario.

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?

La estructura en carpetas hace que el proyecto sea más fácil de navegar, especialmente para nuevos desarrolladores ya que cada carpeta tiene un propósito claro y al momento de agregar en el caso, un nuevo servicio o modelo, no estará todo mezclado, más si estará organizado. Al igual facilita la colaboración y que el proyecto crezca ya que “sus ramas” estarán bien identificadas y organizadas.

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?

Sin una transacción pueden ocurrir inconsistencias si ocurre un error, ya que, como se usan varias consultas SQL, si una falla, todas funcionan como una cascada, entonces si la primera falla, puede que las demás se cumplan pero de manera incorrecta y genere fallos. Se utiliza para verificar medio que, las operaciones dependientes, si la primera se realizó, puede pasar a la segunda y asi sucesivamente, y si una falla, las siguientes no deben ejecutarse.

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 hace para aplicar el patrón de Inyección de dependencia, para que sea más fácil probarlo (hacer test “fakes” sin una base de datos real) y también para que no sea tan difícil probarlo o cambiarlo en el futuro. También el código en si es mas claro, y puede cambiarse el como funciona la DB sin tocar la clase JugadorService. El patrón usado es Inyección de dependencia, que es como “no fabriques tus cosas, mejor que alguien te las de ya listas”

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?

Lo que ocurre es que muestra un NULL, ya que no existe y el read.reader() devuelve un falso. Una alternativa seria mostrar una excepción personalizada con un mensaje mas amigable, ya que seria mas entendible tanto para el programador, ya que sabría si realmente está dando error y no solo se queda en el limbo, y para el usuario.

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?

Debe crearse una tabla amigos donde tenga Jugador id y amigosid representando relaciones bidireccionales. En los servicios se implementa métodos para agregar, eliminar y listar amigos de un jugador y asegura que las relaciones no se repitan ni se agreguen a si mismos como amigos.

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?

En este caso, se establece desde la base de datos, existe una columna llamada FechaCreacion, y registra la fecha exacta en la cual se inserto un nuevo jugador, se maneja con un DATETIME NOT NULL DEFAULT GETDATE(). Sus ventajas son que en el código todo es mas simple, no es necesario escribir un código complejo para ello ya que existe en la DB, y es muy preciso al marcar la fecha y hora, existe organización y exactitud para saber cuándo se agregó.

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?

Se hace por que es mas seguro usar conexiones independientes, a tratar de usar todos, una misma conexión ya que puede causar bloqueos y errores, y asi también se evitan problemas de concurrencia. Las implicaciones podrían ser que se mantiene limpio el ciclo de vida de las conexiones ya que es mas organizado, y se evita bloqueo de conexiones entre operaciones.

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?

Lo que puede pasar es una race condition (La ultima escritura, gana) y puede causar perdida de datos ya que se intentan ingresar al mismo tiempo y puede causar errores

y que los datos se almacenen mal. Se usaría el control de concurrencia, que no permite que dos procesos o mas modifiquen un dato al mismo tiempo y que se generen errores. Considero que sea el mas efectivo, ya que por ejemplo podrían usarse transacciones por lo del efecto cascada, pero aun asi tendríamos un problema muy semejante.

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?

Prácticamente sirve para que se haga un tipo de evaluación sobre si realmente hubo alguna línea afectada en la DB o no, ya que dependiendo de eso, el usuario sabrá si se realizo correctamente o no la consulta y el programa está funcionando como debería. La información que nos muestra es como retroalimentación y nos facilita el diagnostico por si hay errores.

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?

Se podría crear una clase Logger, donde guarde mensajes en un archivo txt y se implementaría en los servicios como una dependencia y afectaría poco la estructura del programa

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?

Se crearía una tabla mundo y otra Jugador mundo para que exista relación muchos a muchos entre jugadores. En el código se agregarían métodos para agregar y eliminar jugadores de mundos al igual que ver de que mundo es cada jugador. Todo eso se adaptaría también en la lógica del inventario si se desea hacer dependiente del mundo.

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

Es una clase en ADO.NET que se usa para establecer y administrar conexión en bases de datos de SQL server. Se usa para administrar la conexión a la base de datos, y se usan métodos tanto para abrir y cerrar una conexión a base de datos. Se puede crear el string de conexión, abrir y cerrar pero también se puede usar un using para evitar que se quede abierta la base de datos, caso falle o de algún error

22. ¿Para qué sirven los SqlParameter?

Sirven para hacer consultas SQL con parámetros, y manejar tipos de datos.