Segundo Trabajo para entregar

Seminario de Lenguajes opción .NET

1er. Semestre - 2023

Importante: Desarrollar con la versión **.NET 7.0**. No deshabilitar **<ImplicitUsings>** ni **<Nullable>** en los archivos de proyecto (**.csproj**). Resolver todos los *warnings* del compilador respecto de las referencias **null**.

Fecha de publicación: 2/6/2023
Fecha límite para la entrega: 19/6/2023

La entrega se realizará por medio de un formulario de Google que se publicará más adelante. Debe subirse la solución completa (habiendo borrado previamente las carpetas /bin y /obj de todos los proyectos) comprimida en un único archivo (preferentemente .zip). El nombre del archivo debe contener los apellidos de los autores del trabajo.

ENUNCIADO

En este trabajo, se requiere aplicar los principios de arquitectura limpia, inversión de dependencias e inyección de dependencias por constructor, siguiendo el estilo de la aplicación desarrollada en la teoría 12.

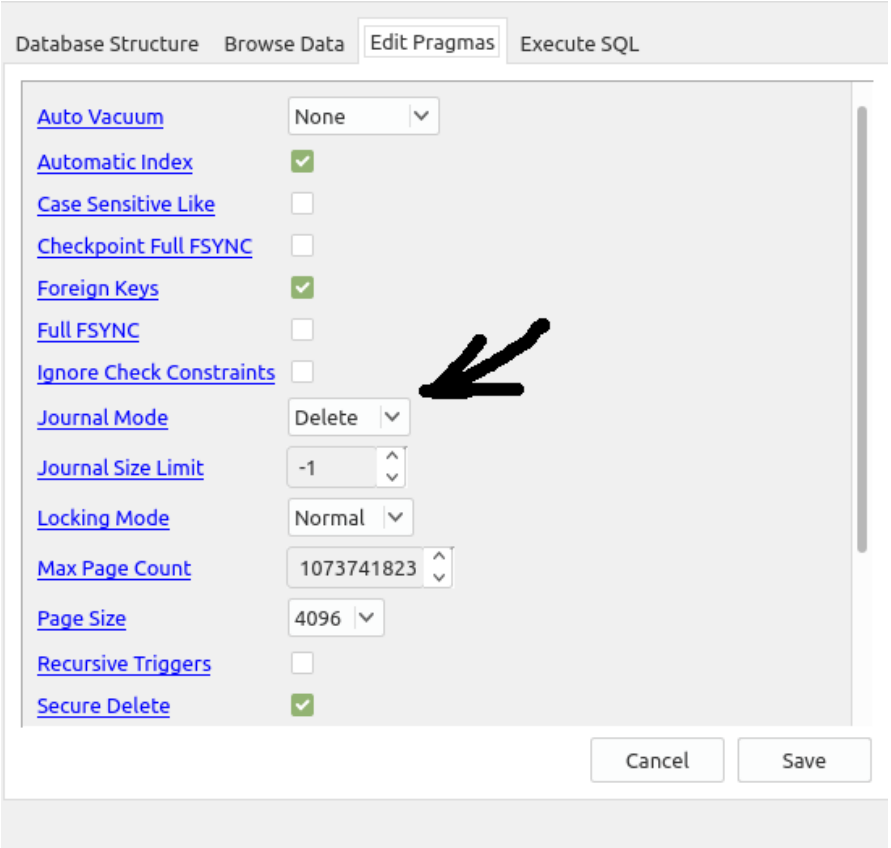
El objetivo de este trabajo es extender el primer proyecto sobre el “SISTEMA PARA LA GESTIÓN DE SEGUROS” realizado anteriormente. Por lo tanto, se debe tomar como base dicho proyecto y considerar las siguientes especificaciones:

1. Completar la funcionalidad para realizar altas, bajas, modificaciones y el listado completo de las entidades '**siniestro**' y '**tercero**'.
2. Al dar de alta un siniestro, se debe validar que el seguro esté vigente. Esto implica que la fecha del siniestro debe estar dentro del periodo de vigencia del seguro. En caso contrario, se debe informar al usuario que no es posible registrar dicho siniestro.
3. En todos los casos, se debe realizar la eliminación en cascada. Es decir, si se elimina un titular, también se deben eliminar sus vehículos. A su vez, la eliminación de los vehículos conlleva la eliminación de las pólizas asociadas y, en caso de existir, la eliminación de los siniestros relacionados. Por último, la eliminación de los siniestros puede implicar la eliminación de terceros, si estos han sido definidos en el siniestro que se está eliminando.
4. En el proyecto "Repositorios", se debe utilizar Entity Framework Core para persistir la información en una base de datos SQLite. Se debe seguir el enfoque “code first”.
5. Se debe definir un repositorio para cada entidad y todos los casos de uso que se consideren necesarios. Cada caso de uso debe ser implementado mediante una clase.

6. La interfaz de usuario debe ser desarrollada utilizando Blazor Server. Diseñar esta interfaz libremente. Tener en cuenta que toda la funcionalidad desarrollada en el primer proyecto tiene ahora que ser accesible desde esta interfaz Blazor.

Nota: Es conveniente establecer la propiedad *journal mode* de la base de datos sqlite en *DELETE.*

Se puede hacer con la aplicación DB Browser for SQLite.



También se puede hacer por código:

```
context.Database.EnsureCreated();
var connection = context.Database.GetDbConnection();
connection.Open();
using (var command = connection.CreateCommand())
{
    command.CommandText = "PRAGMA journal_mode=DELETE;";
    command.ExecuteNonQuery();
}
```

Explicación de esta recomendación

La propiedad *journal_mode* se utiliza en una base de datos SQLite para especificar el modo de registro de transacciones que se utilizará. Éste determina cómo se guardan y administran los cambios realizados en la base de datos.

Al utilizar el modo *DELETE* en SQLite, los cambios realizados se reflejan inmediatamente en la base de datos principal. Esto significa que, después de confirmar una transacción, los datos modificados se guardan directamente en el archivo de la base de datos y están disponibles para su lectura inmediata.

Por otro lado, en otros modos, que implican el uso de registros de transacciones, los cambios no se aplican directamente a la base de datos principal. En cambio, se escriben primero en el archivo de registro de transacciones. Luego, en segundo plano, se realizan las operaciones de fusionar y aplicar los cambios al archivo de la base de datos principal.

Como resultado, si se accede a la base de datos con otra herramienta mientras se utiliza un modo de registro de transacciones, es posible que no se vean los cambios reflejados inmediatamente. Es necesario esperar a que se realice la fusión y aplicación de los registros de transacciones antes de que los cambios sean visibles en la base de datos principal. La frecuencia con la que se realiza este proceso de fusión y aplicación puede variar y depende de factores como la configuración y la carga de trabajo del sistema.

Por lo tanto, el modo *DELETE* proporciona una escritura directa y visible en la base de datos, mientras que otros modos de registro pueden introducir una latencia en la propagación de los cambios debido a las operaciones de fusión y aplicación que deben llevarse a cabo.