tengo una solucion de c# de nombre SistemaManu y un proyecto de nombre SisManu, la aplicación que se quiere hacer es un sistema para una agenda personal, dentro del proyecto tengo dos carpetas una se llama Logica y la Otra se llama Modelo, dentro de carpeta Logica debe estar la clase Conexion.cs para conectarse a la aplicación c# con la base de datos sqlite3 de nombre BDmanu.db la cual tiene una tabla de nombre tb\_contactos (idContactos, Nombre, Telefono, Dirección, Correo y Fechacreacion ), tambien debe estar la clase de nombre LContactos.cs y dentro de esta clase deben estar los métodos para guardar, modificar, listar y eliminar; en la carpeta Modelo debe estar la clase de las propiedades de la tabla tb\_contactos dicha clase se llamara MContactos.cs; podrias darme todos los codigos necesarios para poder realizar este proyecto.

ChatGPT

Claro, puedo proporcionarte una estructura básica del proyecto y los códigos necesarios para comenzar con tu sistema de agenda personal en C# utilizando SQLite como base de datos. Aquí te dejo los archivos que necesitas:

Conexion.cs (en la carpeta Logica):

csharp

Copy code

using System.Data.SQLite;

namespace SistemaManu.Logica

{

public static class Conexion

{

public static SQLiteConnection ObtenerConexion()

{

string cadenaConexion = "Data Source=BDmanu.db;Version=3;";

return new SQLiteConnection(cadenaConexion);

}

}

}

LContactos.cs (en la carpeta Logica):

csharp

Copy code

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SQLite;

using SistemaManu.Modelo;

namespace SistemaManu.Logica

{

public class LContactos

{

public static void GuardarContacto(MContactos contacto)

{

using (var conexion = Conexion.ObtenerConexion())

{

conexion.Open();

string query = "INSERT INTO tb\_contactos (Nombre, Telefono, Direccion, Correo, Fechacreacion) " +

"VALUES (@Nombre, @Telefono, @Direccion, @Correo, @Fechacreacion)";

using (var comando = new SQLiteCommand(query, conexion))

{

comando.Parameters.AddWithValue("@Nombre", contacto.Nombre);

comando.Parameters.AddWithValue("@Telefono", contacto.Telefono);

comando.Parameters.AddWithValue("@Direccion", contacto.Direccion);

comando.Parameters.AddWithValue("@Correo", contacto.Correo);

comando.Parameters.AddWithValue("@Fechacreacion", DateTime.Now);

comando.ExecuteNonQuery();

}

}

}

// Métodos para modificar, listar y eliminar contactos aquí

}

}

MContactos.cs (en la carpeta Modelo):

csharp

Copy code

namespace SistemaManu.Modelo

{

public class MContactos

{

public int IdContacto { get; set; }

public string Nombre { get; set; }

public string Telefono { get; set; }

public string Direccion { get; set; }

public string Correo { get; set; }

public DateTime FechaCreacion { get; set; }

}

}

Estos son los archivos básicos para comenzar tu proyecto de agenda personal en C#. Asegúrate de agregar la referencia al ensamblado System.Data.SQLite a tu proyecto para poder trabajar con SQLite en C#. Además, tendrás que implementar los métodos restantes en la clase LContactos.cs para modificar, listar y eliminar contactos según tus necesidades.

dame todos los metodos necesarios sin obiar ninguno

ChatGPT

¡Por supuesto! Aquí tienes todos los métodos necesarios en la clase LContactos.cs para completar las operaciones básicas de guardar, modificar, listar y eliminar contactos:

csharp

Copy code

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SQLite;

using SistemaManu.Modelo;

namespace SistemaManu.Logica

{

public class LContactos

{

public static void GuardarContacto(MContactos contacto)

{

using (var conexion = Conexion.ObtenerConexion())

{

conexion.Open();

string query = "INSERT INTO tb\_contactos (Nombre, Telefono, Direccion, Correo, Fechacreacion) " +

"VALUES (@Nombre, @Telefono, @Direccion, @Correo, @Fechacreacion)";

using (var comando = new SQLiteCommand(query, conexion))

{

comando.Parameters.AddWithValue("@Nombre", contacto.Nombre);

comando.Parameters.AddWithValue("@Telefono", contacto.Telefono);

comando.Parameters.AddWithValue("@Direccion", contacto.Direccion);

comando.Parameters.AddWithValue("@Correo", contacto.Correo);

comando.Parameters.AddWithValue("@Fechacreacion", DateTime.Now);

comando.ExecuteNonQuery();

}

}

}

public static void ModificarContacto(MContactos contacto)

{

using (var conexion = Conexion.ObtenerConexion())

{

conexion.Open();

string query = "UPDATE tb\_contactos SET Nombre = @Nombre, Telefono = @Telefono, " +

"Direccion = @Direccion, Correo = @Correo WHERE IdContacto = @IdContacto";

using (var comando = new SQLiteCommand(query, conexion))

{

comando.Parameters.AddWithValue("@Nombre", contacto.Nombre);

comando.Parameters.AddWithValue("@Telefono", contacto.Telefono);

comando.Parameters.AddWithValue("@Direccion", contacto.Direccion);

comando.Parameters.AddWithValue("@Correo", contacto.Correo);

comando.Parameters.AddWithValue("@IdContacto", contacto.IdContacto);

comando.ExecuteNonQuery();

}

}

}

public static List<MContactos> ListarContactos()

{

var listaContactos = new List<MContactos>();

using (var conexion = Conexion.ObtenerConexion())

{

conexion.Open();

string query = "SELECT \* FROM tb\_contactos";

using (var comando = new SQLiteCommand(query, conexion))

{

using (var reader = comando.ExecuteReader())

{

while (reader.Read())

{

var contacto = new MContactos

{

IdContacto = Convert.ToInt32(reader["IdContacto"]),

Nombre = reader["Nombre"].ToString(),

Telefono = reader["Telefono"].ToString(),

Direccion = reader["Direccion"].ToString(),

Correo = reader["Correo"].ToString(),

FechaCreacion = Convert.ToDateTime(reader["Fechacreacion"])

};

listaContactos.Add(contacto);

}

}

}

}

return listaContactos;

}

public static void EliminarContacto(int idContacto)

{

using (var conexion = Conexion.ObtenerConexion())

{

conexion.Open();

string query = "DELETE FROM tb\_contactos WHERE IdContacto = @IdContacto";

using (var comando = new SQLiteCommand(query, conexion))

{

comando.Parameters.AddWithValue("@IdContacto", idContacto);

comando.ExecuteNonQuery();

}

}

}

}

}

Con estos métodos completos, ahora puedes realizar todas las operaciones básicas de CRUD (Crear, Leer, Actualizar, Eliminar) en tu aplicación de agenda personal en C# utilizando SQLite como base de datos.

para tener un mejor control de la conexion a la base de datos, podrias darme una clase mas estructurada y robusta

ChatGPT

Claro, puedo proporcionarte una clase de conexión más estructurada y robusta utilizando el patrón Singleton para garantizar una única instancia de la conexión en toda la aplicación y liberar recursos correctamente. Aquí tienes una implementación mejorada de la clase Conexion:

csharp

Copy code

using System;

using System.Data.SQLite;

namespace SistemaManu.Logica

{

public sealed class Conexion

{

private static readonly string cadenaConexion = "Data Source=BDmanu.db;Version=3;";

private static readonly Lazy<SQLiteConnection> instancia = new Lazy<SQLiteConnection>(() => new SQLiteConnection(cadenaConexion));

private Conexion() { }

public static SQLiteConnection ObtenerConexion()

{

return instancia.Value;

}

public static void AbrirConexion()

{

if (instancia.IsValueCreated && instancia.Value.State != System.Data.ConnectionState.Open)

{

instancia.Value.Open();

}

}

public static void CerrarConexion()

{

if (instancia.IsValueCreated && instancia.Value.State != System.Data.ConnectionState.Closed)

{

instancia.Value.Close();

}

}

}

}

Con esta implementación:

La clase Conexion utiliza un campo privado estático instancia para almacenar la conexión SQLite y se inicializa de forma perezosa (lazy) mediante la clase Lazy<T>. Esto garantiza que solo se cree una instancia de SQLiteConnection cuando sea necesaria y se evita la inicialización múltiple.

Se proporcionan métodos estáticos para abrir y cerrar la conexión de manera controlada (AbrirConexion() y CerrarConexion()), lo que facilita el control y la gestión adecuada de la conexión en la aplicación.

La clase se declara como sealed para evitar la herencia y garantizar que no se puedan crear subclases de Conexion, lo que podría romper el patrón Singleton.

Esta implementación mejorada proporciona un mejor control y gestión de la conexión a la base de datos SQLite en tu aplicación C#.

esta clase modifica algo en los metodos detallados?

ChatGPT

La clase Conexion no modifica los métodos detallados en la clase LContactos, pero proporciona una manera más robusta y estructurada de gestionar la conexión a la base de datos SQLite. Los métodos en LContactos permanecen igual en términos de su lógica y funcionalidad.

Aquí están los métodos de LContactos sin cambios:

csharp

Copy code

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SQLite;

using SistemaManu.Modelo;

namespace SistemaManu.Logica

{

public class LContactos

{

public static void GuardarContacto(MContactos contacto)

{

using (var conexion = Conexion.ObtenerConexion())

{

conexion.Open();

string query = "INSERT INTO tb\_contactos (Nombre, Telefono, Direccion, Correo, Fechacreacion) " +

"VALUES (@Nombre, @Telefono, @Direccion, @Correo, @Fechacreacion)";

using (var comando = new SQLiteCommand(query, conexion))

{

comando.Parameters.AddWithValue("@Nombre", contacto.Nombre);

comando.Parameters.AddWithValue("@Telefono", contacto.Telefono);

comando.Parameters.AddWithValue("@Direccion", contacto.Direccion);

comando.Parameters.AddWithValue("@Correo", contacto.Correo);

comando.Parameters.AddWithValue("@Fechacreacion", DateTime.Now);

comando.ExecuteNonQuery();

}

}

}

public static void ModificarContacto(MContactos contacto)

{

using (var conexion = Conexion.ObtenerConexion())

{

conexion.Open();

string query = "UPDATE tb\_contactos SET Nombre = @Nombre, Telefono = @Telefono, " +

"Direccion = @Direccion, Correo = @Correo WHERE IdContacto = @IdContacto";

using (var comando = new SQLiteCommand(query, conexion))

{

comando.Parameters.AddWithValue("@Nombre", contacto.Nombre);

comando.Parameters.AddWithValue("@Telefono", contacto.Telefono);

comando.Parameters.AddWithValue("@Direccion", contacto.Direccion);

comando.Parameters.AddWithValue("@Correo", contacto.Correo);

comando.Parameters.AddWithValue("@IdContacto", contacto.IdContacto);

comando.ExecuteNonQuery();

}

}

}

public static List<MContactos> ListarContactos()

{

var listaContactos = new List<MContactos>();

using (var conexion = Conexion.ObtenerConexion())

{

conexion.Open();

string query = "SELECT \* FROM tb\_contactos";

using (var comando = new SQLiteCommand(query, conexion))

{

using (var reader = comando.ExecuteReader())

{

while (reader.Read())

{

var contacto = new MContactos

{

IdContacto = Convert.ToInt32(reader["IdContacto"]),

Nombre = reader["Nombre"].ToString(),

Telefono = reader["Telefono"].ToString(),

Direccion = reader["Direccion"].ToString(),

Correo = reader["Correo"].ToString(),

FechaCreacion = Convert.ToDateTime(reader["Fechacreacion"])

};

listaContactos.Add(contacto);

}

}

}

}

return listaContactos;

}

public static void EliminarContacto(int idContacto)

{

using (var conexion = Conexion.ObtenerConexion())

{

conexion.Open();

string query = "DELETE FROM tb\_contactos WHERE IdContacto = @IdContacto";

using (var comando = new SQLiteCommand(query, conexion))

{

comando.Parameters.AddWithValue("@IdContacto", idContacto);

comando.ExecuteNonQuery();

}

}

}

}

}

Los cambios en la clase Conexion proporcionan una mejor gestión de la conexión sin afectar la funcionalidad de los métodos en LContactos.