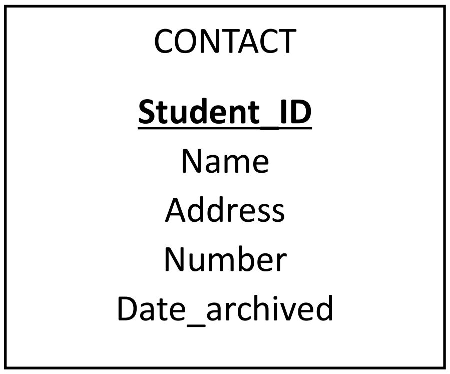
DATABASE MANAGEMENT SYSTEM(PROJECT)

MEMBERS:

Alcanar, France
Dalugdugan,Nicko
Delos Santos, Melchor
Manansala, Richard
Pasamba, Jermaine

ENTITY RELATIONSHIP DIAGRAM(ERD)



C# (code)

(Program.cs)

```
Namespace DBMS_Phonebook;

Using static FileSave;

Using static Utils;

Using MySql.Data.MySqlClient;

Using Spectre.Console;


Class Program

{

   Public const string connectionString = “server=localhost;port=3306;uid=root;database=phonebook;”;
```

```
    Static readonly MySqlConnection connection = new(connectionString);


    Static void Main(string[] args)

    {

        Exception? Error = null;

        AnsiConsole.Status().Start("Connecting to database…", (ctx) =>

        {

            Try

            {

                Using (connection)

                {

                    Connection.Open();

                }

            }

            Catch (Exception err)

            {

                Error = err;

            }

        });

        If (error != null)

        {

            AnsiConsole.MarkupLine("[red]Error connecting to database. Please check your connection
settings.[/]");

            AnsiConsole.MarkupLine("[red]" + error.Message + "[/]");

            AnsiConsole.MarkupLine("\n\nContinuing without database connection may create unexpected
events.");

            Console.ReadKey();

        }
        HomePage();
        AnsiConsole.Markup("\n\n\n Good Bye!");
    }



    Static void HomePage()
    {
        Bool loop = true;
        Do
        {
            Int a = InteractiveInput(
            "PhoneBook \n \nWhat would you like to do?",
            New string[] { "View All Contacts", "Find contact", "Add Contact", "Archive a Contact", "Archive All",
"Show History", "Quit" });
            Switch (a)
            {
                Case 0:
                    ViewAllContactsPage();
                    Break;
                Case 1:
                    FindContactPage();
                    Break;
                Case 2:
                    AddContactPage();
                    Break;
                Case 3:
                    ArchiveContactPage();
                    Break;
                Case 4:
```

```
                ArchiveAllPage();
                Break;
            Case 5:
                ShowHistoryPage();
                Break;
            Default:
                Loop = false;
                Break;
        }
    } while (loop);
}
Static void ArchiveAllPage()
{
    ClearScreen();
    Var allContacts = RetrieveData(connection);
    AnsiConsole.MarkupLine("[red]Archive all?[/]" + "\n");
    If (allContacts.Count > 0)
    {
        Int res = InteractiveInput("Are you sure you want to archive everything?", new string[] { "No",
"Yes" });
        AnsiConsole.MarkupLine("\n");
        If (res == 1)
        {
            ArchiveAll(connection);
            ACout("Archive succesfull!");
        }
        Else
        {
            ACout("Archive aborted.");
        }
    }
    Else
    {
        ACout(“No contacts to archive!”);
    }
    Pause();
}

Public static void ShowHistoryPage()
{
    ClearScreen();
    AnsiConsole.Markup(“Archived History\n\n”);
    Var allContacts = GetHistory(connection);
    If (allContacts.Count == 0)
    {
        ACout(“Empty…. ☹ \n”);
    }
    Else
    {
        Var table = new Table().AddColumns(“Date Archived”, “Name”, “Number”, “Address”);
        Foreach (Contact h in allContacts)
        {
            Table.AddRow(h.dateArchived!, h.name, h.number, h.address);
        }
        AnsiConsole.Write(table);
    }
    Pause();
}
```

```
Public static void ViewAllContactsPage()
{
   Var allContacts = RetrieveData(connection);
   ClearScreen();
   AnsiConsole.Markup(“All contacts list \n\n”);
   If (allContacts.Count == 0)
   {
      ACout(“No contacts to show ☹ \n”);
   }
   allContacts.Sort((x, y) => x.name.CompareTo(y.name));
   int index = 0;
   var table = CreateContactTable();
   foreach (Contact h in allContacts)
   {
      AnsiConsole.Clear();
      Table.AddRow((index + 1).ToString(), h.name, h.number, h.address);
      Index++;
      AnsiConsole.Write(table);
      Thread.Sleep(70);
   }
   Pause();
}

Public static void AddContactPage()
{
   ClearScreen();
   AnsiConsole.Markup(“Create a new contact\n\n”);
   Contact newContact = new()
```

```
    {
      Name = AnsiConsole.Ask<string>("[green]Enter name: [/]"),
      Number = AnsiConsole.Prompt(new TextPrompt<string>("[green]Enter Number:
[/]").ValidationErrorMessage("[red]Invalid phone number.[/]").Validate(IsValidPhoneNumber)),
      Address = AnsiConsole.Ask<string>("[green]Enter Address: [/]")
    };
    AnsiConsole.MarkupLine("\nAdding to database....");

    newContact.TrimMembers();

    using (connection)
    {
      Try
      {
        Connection.Open();
        Var command = connection.CreateCommand();
        Command.CommandText = "INSERT INTO contact (name, number, address) VALUES (@name,
@number, @address)";
        Command.Parameters.AddWithValue("@name", newContact.name);
        Command.Parameters.AddWithValue("@number", newContact.number);
        Command.Parameters.AddWithValue("@address", newContact.address);
        Command.ExecuteNonQuery();
      }
      Catch (MySqlException)
      {
        AnsiConsole.MarkupLine($"Error inserting contact. Number: ({newContact.number}) already in
used. ");
        HomePage();
      }
    }
    AnsiConsole.MarkupLine(“[green]Added sucessfully![/]”);


    Pause();
}


Public static void FindContactPage()
{
    ClearScreen();
    AnsiConsole.Markup(“Find a contact\n\n”);
    String name = “”;
    List<Contact> contactsFound = new();
    Var allContacts = RetrieveData(connection);
    Name = AnsiConsole.Ask<string>(“[green]Enter name: [/]”);
    Console.WriteLine();
    Foreach (Contact c in allContacts)
    {
        If (c.name == name)
        {
            contactsFound.Add©;
        }
    }


    If (contactsFound.Count > 0)
    {
        Var table = CreateContactTable(“Name”, “Number”, “Address”);
        AnsiConsole.Markup(“Here are the contacts with the name: “ + name + “\n\n”);
        Foreach (Contact c in contactsFound)
        {
            Table.AddRow(c.name, c.number, c.address);
        }
        AnsiConsole.Write(table);
    }
    Else
    {
        ACout("No contact found with the name. ☹");
    }
    Thread.Sleep(300);
    Pause();
}

Public static void ArchiveContactPage()
{
    ClearScreen();
    AnsiConsole.Markup("Archive a contact\n\n");
    String name = AnsiConsole.Ask<string>("Enter name: ");
    List<Contact> contactsFound = new();
    Try
    {
        contactsFound = GetContact(connection, name);
    }
    Catch (Exception err)
    {
        AnsiConsole.MarkupLine(err.Message);
        Console.ReadKey();
    }

    If (contactsFound.Count <= 0)
    {
```

```
  ACout(“No contact found with the name. ☹”);
}
Else
{
  Switch (contactsFound.Count)
  {
    Case 1:
      {
        Contact n = contactsFound[0];
        AnsiConsole.MarkupLine(n.ToString());
        Int res = InteractiveInput(
          “Are you sure you want to archive? \n\n” + n,
          New string[] { “Back”, “Archive” });
        If (res == 1)
        {
          ArchiveContact(connection, n.id);
          ACout(“Contact archived!”);
        }

        Break;
      }
    Default:
      {
        List<string> choices = new() { “Back” };
        Foreach (Contact c in contactsFound)
        {
          Choices.Add(c.ToString());
        }
```

```
                Int res = InteractiveInput("Select contact to archived: ", choices.ToArray());
                Switch (res)
                {
                    Case 0:
                        Break;
                    Default:
                        Contact n = contactsFound[res – 1];
                        AnsiConsole.Markup(n.ToString());
                        Int res1 = InteractiveInput(
                            "Are you sure you want to archive? \n\n" + n, new string[]
                            {"Back", "Archive"});
                        If (res1 == 1)
                        {
                            ArchiveContact(connection, n.id);
                            ACout("Contact archived!");
                            Console.WriteLine();
                        }
                        Break;
                }

                Break;
            }
        }
    }

    Console.ReadKey();
}
```

```
}
C#
(utils.cs)
Namespace DBMS_Phonebook;
Using System;
Using Spectre.Console;

Static class Utils
{

   Public static void ClearScreen()
   {
      Console.Clear();

   }
   Public static void ACout(string h, int delay = 50)
   {
      Int index = 0;
      Foreach (char c in h)
      {
         If (Console.KeyAvailable)
         {
            Console.ReadKey(true);
            Console.Write(h.Substring(index));
            Break;
         }
         Console.Write©;
         Index++;
         Thread.Sleep(delay);
```

```
        }

    }


    Public static void Pause()

    {

        Console.WriteLine("\n\nPress any key to continue…");

        Console.ReadKey();

    }

    Public static int InteractiveInput(string label, string[] choices, string endLabel = "Press ENTER/SPACE to
select. Arrow keys to move. \n")

    {

        Int _index = 0;

        Bool looping = true;

        Int choiceIndex = 0;

        Int maxNum = choices.Length – 1;


        ClearScreen();

        ACout(label + "\n\n");

        Thread.Sleep(75);

        Foreach (string h in choices)

        {

            Var s = "";

            If (_index == choiceIndex)

            {

                S += "[green]=>";

            }

            Else

            {

                S += "  [default]";
            }
            S += h + "[/]";

            AnsiConsole.Markup(s);
            Console.Write("\n");
            choiceIndex++;
            Thread.Sleep(75);
        }
        ACout("\n" + endLabel);
        While (looping)
        {
            choiceIndex = 0;
            ClearScreen();
            Console.WriteLine(label + "\n");
            Foreach (string h in choices)
            {
                Var s = "";
                If (_index == choiceIndex)
                {
                    S += "[green]=>";
                }
                Else
                {
                    S += "  [default]";
                }
                S += h + "[/]";

                AnsiConsole.Markup(s);
                Console.Write("\n");
                choiceIndex++;
            }
            Console.Write("\n" + endLabel);

            Switch (Console.ReadKey().Key)
            {
                Case ConsoleKey.UpArrow:
                Case ConsoleKey.W:
                    _index--;
                    If (_index < 0)
                    {
                        _index = maxNum;
                    }
                    Break;
                Case ConsoleKey.DownArrow:
                Case ConsoleKey.S:
                    _index++;
                    If (_index > maxNum)
                    {
                        _index = 0;
                    }
                    Break;
                Case ConsoleKey.Enter:
                Case ConsoleKey.Spacebar:
                    Looping = false;
                    Break;
            }
        }
        Return _index;
```

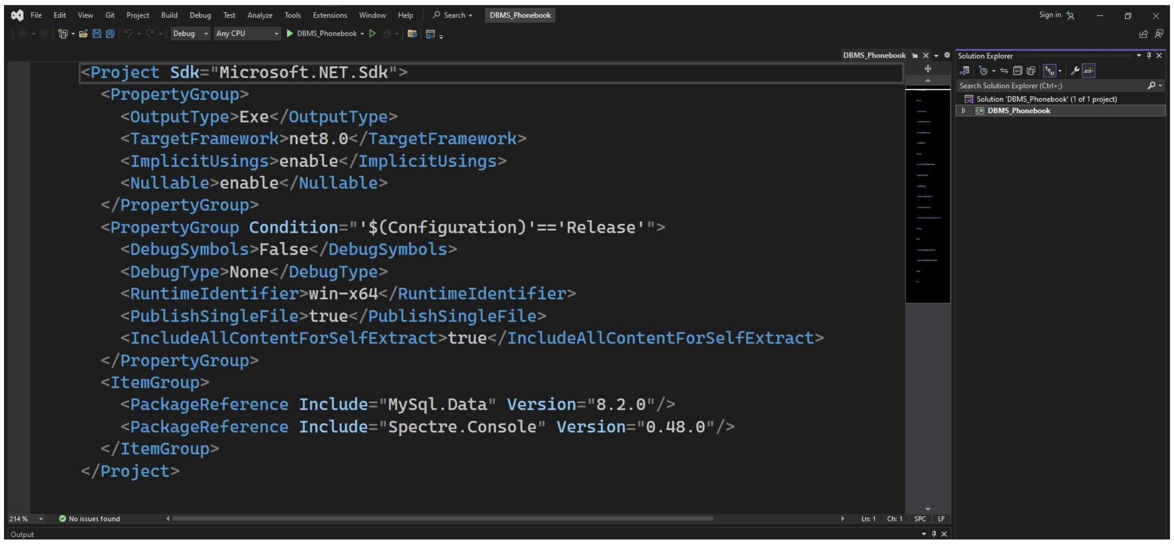
```
    }

    Public static string GetLine(string label, Func<string, bool>? Invalidator = null)
    {
      Invalidator ??= IsEmpty;
      String? Input;
      Do
      {
        ACout(label);
        Input = Console.ReadLine();
      } while (invalidator(input!));
      Return input!;
    }

    Public static bool IsEmpty(string h)
    {
      Return h == string.Empty || h.Trim() == string.Empty;
    }

    Public static bool IsValidPhoneNumber(string phoneNumber)
    {
      // Check if the string contains only valid characters
      Foreach (char ch in phoneNumber)
      {
        If (!int.TryParse(ch.ToString(), out int a) && ch != '-' && ch != '(' && ch != ')' && ch != ' ' && ch !=
  '+')
          Return false;
      }
```

```
        // Count the number of digits in the string
        Int digitCount = 0;
        Foreach (char ch in phoneNumber)
        {
            If (int.TryParse(ch.ToString(), out int a))
                digitCount++;
        }

        // Check if the phone number has at least 10 digits
        If (digitCount < 5)
            Return false;

        // The phone number is valid
        Return true;
    }
    Public static bool IsNotValidPhoneNumber(string phoneNumber)
    {
        Return !IsValidPhoneNumber(phoneNumber);
    }

    Public static Table CreateContactTable(params string[] columns) => new
Table().AddColumns(columns.Length > 0 ? columns : new string[] { "Index", "Name", "Number",
"Address" });




}

Public struct Contact
```

```
{
    Public int id;
    Public string number;
    Public string name;
    Public string address;

    Public string? dateArchived;

    Public Contact(string number, string name, string address, int id, string? archivedAt = null)
    {
        This.number = number;
        This.name = name;
        This.address = address;
        This.id = id;
        This.dateArchived = archivedAt;
    }

    Public override string ToString()
    {
        String n = "\n";
        Return "Name: " + name + n + "Number: " + number + n + "Address: " + address + n;
    }

    Public void TrimMembers()
    {
        Number = number.Trim();
        Name = name.Trim();
        Address = address.Trim();
    }
```

```
};
C#
(filesave.cs 2)
Namespace DBMS_Phonebook;
Using System;
Using MySql.Data.MySqlClient;
Using static Utils;


Public static class FileSave
{

   Public static void SaveToFile(List<Contact> allContacts)
   {
      Using (StreamWriter writer = new StreamWriter("contacts_save.txt"))
      {
         Foreach (Contact c in allContacts)
         {
            Writer.WriteLine(c.name);
            Writer.WriteLine(c.number);
            Writer.WriteLine(c.address);
            Writer.WriteLine("\n");
         }
      }
   }

   Public static void ArchiveContact(MySqlConnection connection, int id)
   {
      Using (connection)
```

```
        {
          Try
          {
            Connection.Open();
            Var command = connection.CreateCommand();
            Command.CommandText = “UPDATE contact SET dateArchived = NOW() WHERE id = @id”;
            Command.Parameters.AddWithValue(“@id”, id);
            Command.ExecuteNonQuery();
          }
          Catch (System.Exception err)
          {
            Console.WriteLine(err.Message);
          }
        }
      }

      Public static void ArchiveAll(MySqlConnection connection)
      {
        Using (connection)
        {
          Try
          {
            Connection.Open();
            Var command = connection.CreateCommand();
            Command.CommandText = “UPDATE contact SET dateArchived = NOW() WHERE dateArchived IS
NULL”;
            Command.ExecuteNonQuery();
          }
          Catch (System.Exception err)
          {
             Console.WriteLine(err.Message);
          }
       }
    }
    Public static List<Contact> GetContact(MySqlConnection connection, string name)
    {
       List<Contact> contactsFound = new();
       Using (connection)
       {
          Connection.Open();
          Var command = connection.CreateCommand();
          Command.CommandText = "SELECT * FROM contact WHERE name = @name AND dateArchived IS
NULL";
          Command.Parameters.AddWithValue("@name", name);
          Var reader = command.ExecuteReader();
          While (reader.Read())
          {
             Contact c = new()
             {
                Id = (int)reader["id"],
                Name = reader["name"].ToString()!,
                Number = reader["number"].ToString()!,
                Address = reader["address"].ToString()!
             };
             contactsFound.Add©;
          }
       }
       Return contactsFound;
    }

    Public static List<Contact> GetHistory(MySqlConnection connection)
    {
        List<Contact> contactsFound = new();
        Using (connection)
        {
            Try
            {
                Connection.Open();
                Var command = connection.CreateCommand();
                Command.CommandText = “SELECT * FROM contact WHERE dateArchived IS NOT NULL ORDER
BY dateArchived DESC”;
                Var reader = command.ExecuteReader();
                While (reader.Read())
                {
                    Contact c = new()
                    {
                        Id = (int)reader[“id”],
                        Name = reader[“name”].ToString()!,
                        Number = reader[“number”].ToString()!,
                        Address = reader[“address”].ToString()!,
                        dateArchived = reader[“dateArchived”].ToString()!
                    };
                    contactsFound.Add©;
                }
            }
            Catch (System.Exception err)
        {
            Console.WriteLine(err);
        }
    }
    Return contactsFound;
}

Public static List<Contact> RetrieveData(MySqlConnection connection)
{
    List<Contact> allContacts = new();
    Using (connection)
    {
        Try
        {
            Connection.Open();
            Var command = connection.CreateCommand();
            Command.CommandText = “SELECT * FROM contact WHERE dateArchived IS NULL”;
            Var reader = command.ExecuteReader();

            While (reader.Read())
            {
                Contact c = new()
                {
                    Id = (int)reader[“id”],
                    Name = reader[“name”].ToString()!,
                    Number = reader[“number”].ToString()!,
                    Address = reader[“address”].ToString()!
                };
                allContacts.Add©;
            }




         }
         Catch (Exception er)
         {
            Console.WriteLine("Error connecting to database." + er.Message);
         }
      }
      Return allContacts;
   }
}
```



```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>
  <PropertyGroup Condition="'$(Configuration)'=='Release'">
    <DebugSymbols>False</DebugSymbols>
    <DebugType>None</DebugType>
    <RuntimeIdentifier>win-x64</RuntimeIdentifier>
    <PublishSingleFile>true</PublishSingleFile>
    <IncludeAllContentForSelfExtract>true</IncludeAllContentForSelfExtract>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="MySql.Data" Version="8.2.0"/>
    <PackageReference Include="Spectre.Console" Version="0.48.0"/>
  </ItemGroup>
</Project>
```

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
DBMS_Phonebook
Sign in
Debug
Any CPU
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'DBMS_Phonebook' (1 of 1 project)
DBMS_Phonebook
214 %
No issues found
Ln: 1 Ch: 1 SPC LF
Output

MYSQL

<Project Sdk="Microsoft.NET.Sdk">

```
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>
  <PropertyGroup Condition="'$(Configuration)'=='Release'">
    <DebugSymbols>False</DebugSymbols>
    <DebugType>None</DebugType>
    <RuntimeIdentifier>win-x64</RuntimeIdentifier>
    <PublishSingleFile>true</PublishSingleFile>
    <IncludeAllContentForSelfExtract>true</IncludeAllContentForSelfExtract>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="MySql.Data" Version="8.2.0"/>
    <PackageReference Include="Spectre.Console" Version="0.48.0"/>
  </ItemGroup>
</Project>
```