

ЛАБОРАТОРНА РОБОТА 10

Тема: Створення графічного додатку типу WindowsForm для роботи зі зв'язаними таблицями бази даних

Мета роботи: набути практичних навиків програмної обробки записів бази даних.

Методичне, програмне забезпечення і ТЗН: Інструкція до лабораторної роботи №1, персональний комп'ютер, ОС Windows 7/8.1/10, Microsoft Office 2010-2019, Adobe Reader, Visual Studio 2019.

Короткі теоретичні відомості

1 Архітектура даних ADO.NET (ActiveX Data Objects)

У Visual Studio .NET є безліч вбудованих майстрів і дизайнерів, які допоможуть швидко і ефективно створити архітектуру доступу до даних під час розробки та оснастити додаток надійним механізмом доступу до даних, витративши мінімум зусиль на написання коду. Поряд з цим всі можливості об'єктної моделі ADO.NET доступні програмно, що дозволяє реалізувати не- стандартні функції або створювати додатки, орієнтовані на потреби користувача.

Доступ до даних в ADO.NET заснований на використанні двох компонентів:

- *провайдера даних* (Data Provider), що виконує функції посередника при взаємодії програми і баз даних;
- *набору даних* - об'єкта DataSet, в якому дані зберігаються на локальному комп'ютері.

2 Провайдери даних

Зв'язок з базою даних створюється і підтримується за допомогою провайдера даних.

Провайдерами даних .NET Framework є компоненти, які спеціально сконструйовані для обробки даних і швидкого, однопрохідного дос тупу до даних тільки для читання. Насправді *провайдер* - це набір класів, взаємопов'язаних компонентів, що забезпечують ефективний високо продуктивний доступ до даних. Будь-який провайдер даних складається з чотирьох компонентів:

- **Connection**- забезпечує підключення до джерела даних;
- **Command**- застосовується для управління джерелом даних; дозволяє виконувати команди, що не повертають даних, наприклад INSERT, UPDATE і DELETE, або команди, які повертають об'єкт DataReader (такі як SELECT). Об'єкт

Command дозволяє звертатися до команд бази даних для повернення даних, зміни даних, виконання збережених процедур і передачі або отримання відомостей про параметри;

- **DataReader**- надає доступний тільки для односпрямованого читання набору записів, підключеного до джерела даних. DataReader забезпечує високопродуктивний потік даних з джерела даних;

- **DataAdapter** - заповнює від'єднаний об'єкт **DataSet** або **DataTable** і оновлює його вміст. DataAdapter надає міст між об'єктом DataSet і джерелом даних. DataAdapter використовує об'єкти Command для виконання команд SQL на джерелі даних, для завантаження DataSet з даними і узгодження змін даних, виконаних в DataSet , знову з джерелом даних.

Об'єкт Connection представляє з'єднання з базою даних. Visual Studio підтримувати два класи Connection:

- SqlConnection, призначений для підключення до SQL Server;
- OleDbConnection, який застосовується для підключення до різних БД.

Всі дані, необхідні для відкриття каналу зв'язку з базою даних, зберігаються у властивості **ConnectionString** об'єкта Connection, цей об'єкт також підтримує ряд методів, що дозволяють обробляти дані з застосуванням транзакцій.

Об'єкт Command також представлений двома класами – SqlCommand і OleDbCommand. Він дозволяє виконувати команди над базою даних, використовуючи для обміну даними встановлене з'єднання.

За допомогою об'єктів **Command** можна виконувати збережені процедури, команди SQL, а також оператори, які повертають цілі таблиці.

Об'єкт **SqlDataReader** надає потік з набором записів бази даних, доступний тільки для односпрямованого читання.

На відміну від інших компонентів провайдера даних, створювати екземпляри DataReader безпосередньо не дозволяється, його можна отримати за допомогою методів **ExecuteReader** об'єкта Command: метод **SqlCommand.ExecuteReader()** повертає об'єкт **SqlDataReader**. Оскільки в будь-який момент часу в пам'яті знаходиться тільки один рядок, використання об'єкта SqlDataReader майже не знижує продуктивність системи, але вимагає монопольного доступу до відкритого об'єкту Connection протягом часу життя об'єкта SqlDataReader.

3 Організація доступу до даних

Доступ до даних в ADO.NET здійснюється в такій послідовності:

1. Об'єкт Connection встановлює між додатком і базою даних з'єднання, безпосередньо доступне для об'єктів Command і DataAdapter.

2. Об'єкт `Command` дозволяє виконувати команди безпосередньо над базою даних. Якщо виконана команда повертає кілька значень, `Command` відкриє до них доступ через об'єкт `DataReader`.

3. Отримані результати можна обробляти безпосередньо, використовуючи код програмно. Для оновлення бази даних також застосовують об'єкти `Command`.

4 Підключення до бази даних

Об'єкт `Connection`

Створити об'єкт `Connection` в коді програми досить просто. Для цього треба створити новий об'єкт `Connection` і привласнити його властивості `ConnectionString` рядок з'єднання з базою даних, як показано в лістингу нижче.

Приклад. Створення об'єкта `Connection` в коді програми

```
SqlCeEngine^ dbEngine = gcnew SqlCeEngine();
dbEngine->LocalConnectionString = "Data Source=PfoneDB.sdf";
connect = gcnew SqlConnection(dbEngine->LocalConnectionString);
```

5 Об'єкт `Command`

Об'єкт `Command` містить посилання на збережену процедуру БД або оператор SQL і здатний виконати цей оператор на джерелі даних, використовуючи активне з'єднання. Об'єкт `Command` також містить всі дані, необхідні для виконання команди: посилання на активну команду, текст команди і її параметри.

Об'єкти `Command` дуже швидко і ефективно взаємодіють з базами даних, дозволяючи виконувати різні операції:

- виконувати команди, що не повертають значення, наприклад `INSERT`, `UPDATE` і `DELETE`;
- виконувати команди, які повертають єдине значення;
- виконувати команди мови `Database Definition Language`, наприклад `CREATE TABLE` і `ALTER`;
- повертати результуючий набір безпосередньо через екземпляр об'єкта `DataReader` - це найшвидший спосіб доступу до даних, він особливо зручний, якщо дані потрібні тільки для читання;
- повертати результуючий набір у вигляді потоку XML - цю можливість підтримувати тільки клас `SqlCommand`;
- повертати результуючий набір, створений на основі кількох таблиць або в результаті виконання декількох операторів.

У об'єкта `Command` є властивість **CommandType**, яке визначає тип команди, що міститься у властивості **CommandText**, воно може приймати одне з наступних значень:

- **Text** - змушує розглядати значення властивості `CommandText` як текст команди SQL, при цьому в властивості `CommandText` повинен бути один або кілька допустимих операторів SQL, між якими ставиться крапка з комою. В останньому випадку оператори SQL виконуються по порядку;
- **StoredProcedure** - в цьому випадку у властивості `CommandText` необхідно вказати ім'я існуючої процедури, - вона буде виконана при виклику даної команди;
- **TableDirect** - в цьому випадку у властивості `CommandText` має бути ім'я однієї або кількох таблиць. При виконанні ця команда поверне всі стовпці і рядки таблиць, заданих властивістю `CommandText`.

Наприклад:

```
SqlCommand cmd = new SqlCommand();  
cmd.Connection = con;  
cmd.CommandType = CommandType.Text;  
cmd.CommandText = "SELECT * FROM Student";
```

Властивість `Connection` встановлюється відповідно до типу з'єднання: для об'єкта `SqlCommand` потрібно з'єднання на основі об'єкта `SqlConnection`, а для `OleDbCommand` - з'єднання на основі `OleDbConnection`.

Об'єкт `Command` підтримує три методи:

- **ExecuteNonQuery()** - виконує команди, що не повертають дані, наприклад `INSERT`, `UPDATE` і `DELETE`;
- **ExecuteScalar()** - виконує запити до БД, які повертають єдине значення;
- **ExecuteReader()** - повертає результуючий набір через об'єкт `DataReader`.

Об'єкт *DataReader*

Виконати за допомогою об'єкта `Command` команду, яка повертає набір, можна методом `ExecuteReader()`. Він повертає об'єкт `DataReader`.

Об'єкт `DataReader`- це спрощений об'єкт, що забезпечує швидке і ефективно послідовне односпрямоване читання даних. Об'єкт `DataReader` дозволяє перебирати записи результуючого набору, передаючи потрібні значення безпосередньо коду програми. При цьому дані дозволяється переглядати тільки в одному напрямку: не можна повернутися до запису, прочитаної раніше. Крім того, об'єкт `DataReader`

орієнтований на використання постійних з'єднань і, поки він існує, вимагає монопольного доступу до активного з'єднання.

Об'єкт `DataReader` не можна створити безпосередньо, це робиться шляхом виклику методу `ExecuteReader` об'єкта `Command`. Подібно іншим членам класів провайдерів даних, у кожного класу `DataProvider` є власний клас `DataReader`. Наприклад, об'єкт `SqlCommand` повертає `SqlDataReader`.

При виклику методу `ExecuteReader` об'єкт `Command` виконує подану ним команду і створює об'єкт `DataReader` відповідного типу, який можна записати в змінну посилального типу.

Отримавши посилання на об'єкт `DataReader`, можна переглядати записи, завантажуючи потрібні дані в пам'ять. У нового об'єкта `DataReader` показчик читання встановлюється на перший запис результуючого набору. Щоб зробити її доступною, слід викликати метод **Read**. Якщо запис доступний, метод `Read` переводить показчик об'єкта `DataReader` до наступного запису і повертає `true`, в іншому випадку метод `Read` повертає `false`. Таким чином, метод `Read` використовують для перебору записів в циклі `while`, наприклад, так:

```
SqlCeDataReader^ sqlRead = command->ExecuteReader();
while (sqlRead->Read())
{
    ...
}
```

Виконання методу `ExecuteReader()` вимагає відкритого з'єднання з базою даних. Закінчивши читання даних, слід викликати метод `Close()` об'єкта `DataReader`, щоб закрити з'єднання. При читанні записів за допомогою об'єкта `DataReader` значення окремих полів доступні через індексатор або властивість за замовчуванням у вигляді масиву об'єктів, до елементів якого дозволяється звертатися за індексом:

```
void ExecuteReader (String^ query){
    connect->Open();
    SqlCommand^ command = gcnew SqlCommand(query, connect);
    command->ExecuteNonQuery();
    connect->Close();
}
```

Клас `MessageBox`

У класі **`MessageBox`** визначено набір методів, з яких найчастіше використовується статичний перевантажений метод **`MessageBox::Show()`**, що забезпечує відображення діалогового вікна. Цей метод може бути викликана з

різною кількістю аргументів. Наприклад, типовий список аргументів визначається таким прототипом:

```
static DialogResult Show(  
    String^ text, // Текст в клієнтській області вікна  
    String^ caption, // Текст в заголовку вікна  
    MessageBoxButtons buttons, // Кнопки в клієнтській області вікна  
    MessageBoxIcon icon, / Піктограма в клієнтській області вікна  
    MessageBoxDefaultButton defaultButton // Кнопка, що обирається, за  
                                           // замовчуванням  
)
```

З прототипу видно, що метод `MessageBox::Show()` також, як і метод `ShowDialog()`, після свого виконання повертає значення, яке дозволяє програмно визначити, якою кнопкою користувач закриватиме діалогове вікно, і вирішити, як повинна виконуватися програма далі. Обробники подій, пов'язаних з натисканням кнопок діалогового вікна, реалізовані в самому класі `MessageBox`. Доступні в клієнтській області діалогового вікна кнопки, призначені для закриття вікна, задаються константою перелічуваного типу:

- **`MessageBoxButtons::AbortRetryIgnore`** – Стоп/Повтор/Пропустити;
- **`MessageBoxButtons::OK`** – ОК;
- **`MessageBoxButtons::OKCancel`** – ОК/Скасування;
- **`MessageBoxButtons::RetryCancel`** – Повтор/Скасування;
- **`MessageBoxButtons::YesNo`** – Та/ні;
- **`MessageBoxButtons::YesNoCancel`** - Так/Ні/Скасування

Піктограма, що інформує користувача про характер виведеного тексту (нагадування, попередження, заборона і т.д.), задається константою перелічуваного типу:

- **`MessageBoxIcon::Information`** – інформування;
- **`MessageBoxIcon::Error`** – помилка;
- **`MessageBoxIcon::Hand`** – заборона;
- **`MessageBoxIcon::Stop`** – стоп;
- **`MessageBoxIcon::Warning`** – попередження;
- **`MessageBoxIcon::Question`** – очікування відповіді;
- **`MessageBoxIcon::None`** – відсутність піктограми).

У діалоговому вікні за допомогою константи перелічуваного типу `MessageBoxDefaultButton` можна вказати кнопку, яка буде закривати вікно натисканням клавіші `Enter`, якщо користувач не бажає вибирати одну з наявних у вікні кнопок за допомогою миші або клавіші `Tab`.

Порядок виконання роботи

1. Завантажити Microsoft Visual Studio 2019.
2. Створити новий проект Windows Forms додатки (платформа .NET Framework)
3. Додати стартову форму, що буде завантажуватись першою при запуску програми: у вікні Solution explorer викликати контекстне меню для назви проекту і вибрати пункт **Добавить – Создать элемент – Форма Windows Forms** і ввести ім'я **MyForm**.
4. Створити інтерфейс головної форми, змінивши властивості text та image згідно рисунку 1 та розмістивши на ній наступні компоненти: button 1 і button2 для завершення та початку роботи програми відповідно, menuStrip1 для створення головного меню, toolStrip1 для створення панелі інструментів та toolTip1 для створення впливаючих підказок при наведенні на кнопки button1 і button2 (рисунок 1).



Рисунок 1 – Вигляд стартової форми у режимі конструктора

5. Створити команди пункту головного меню «Файл» (рисунок 2):



Рисунок 2 – Команди пункту головного меню «Файл»

6. Створити команди пункту головного меню «Вид» (рисунок 3):




Рисунок 3 – Команди пункту головного меню «Вид»

7. Створити команду пункту головного меню «Довідка» (рисунок 4):




Рисунок 4 – Команди пункту головного меню «Довідка»


8. Створити кнопку панелі інструментів для виведення довідкової інформації про програму, обравши з випадаючого списку елементів  компоненти **toolStrip1** елемент **button** та змінивши його властивість **Image** для зміни вигляду кнопки та візуальної інформативності її призначення згідно наведеного вище рисунку 4.

9. Повторити ті ж дії для створення кнопки на панелі інструментів для завершення роботи програми.

10. Змінити властивість **Image** кнопок **button 1** та **button2** згідно наведених вище рисунків.

11. Запрограмувати кнопку **button1** () , двічі клікнувши по ній та ввівши наступний код:

```
this->Close();
```

12. Запрограмувати та кнопку панелі інструментів . Для цього перейдіть у вікно Properties→Events→Click→button1_Click(рисунок 5). Аналогічно приєднати метод пункту головного меню **Файл – Exit**.

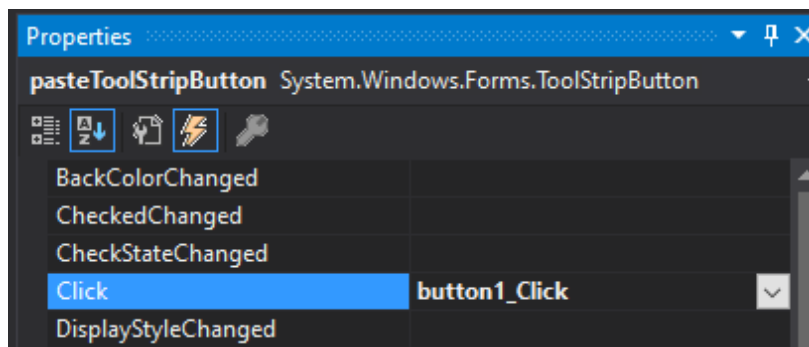



Рисунок 5 – Приклад приєднання події кнопки панелі інструментів до методу командної кнопки button1

13. Забезпечити виведення підказок при наведенні на відповідні кнопки стартової форми програми. Для цього спочатку обрати кнопку  і у вікні властивостей **Properties** у значення властивості **ToolTip** на **ToolTip1** ввести **Вийти з програми**. Аналогічні дії виконати для кнопки, виділеної на рисунку 6:

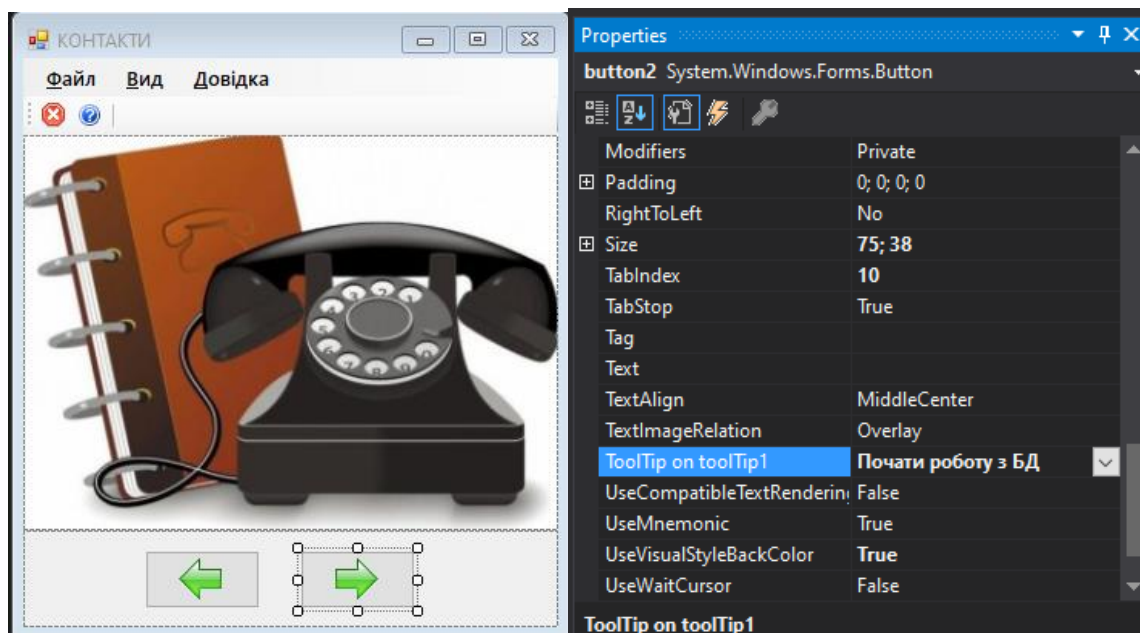


Рисунок 6 – Приклад створення впливаючої підказки

14. Додати форму, що буде завантажуватись при виборі пункту головного меню **Довідка** → **About**: у вікні **Solution explorer** викликати контекстне меню для назви проекту і вибрати пункт **Добавить – Создать элемент – Форма Windows Forms** і ввести ім'я **About**.

15. Розмістити на даній формі елемент керування **Label1** (рисунок 7) та змінити його властивість **text** згідно наведеного нижче рисунку:

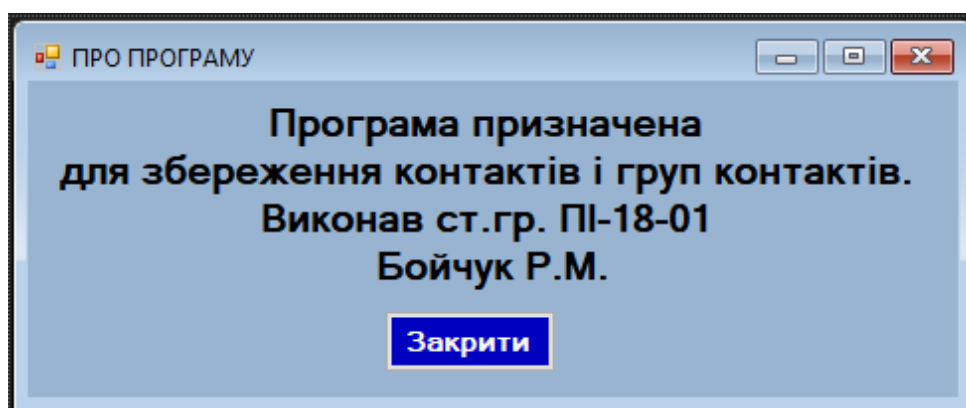



Рисунок 7 – Вигляд вікна форми About у режимі конструктора

16. Запрограмувати кнопку панелі інструментів для виведення довідкової інформації про програму , двічі клікнувши по ній:

```
About^ gr = gcnw About();
gr->Show();
```

17. Приєднати метод пункту головного меню **Довідка** → **About** до методу кнопку панелі інструментів «Про програму».

18. Запрограмувати команду головного меню **Вид** → **Панель інструментів** для відображення чи приховування панелі інструментів, двічі клікнувши по ній та ввівши у заготовці процедури обробки даного пункту наступний код:

```
toolStrip1->Visible = !toolStrip1->Visible;
optionsToolStripMenuItem->Checked = !optionsToolStripMenuItem->Checked;
```

19. Інтегрувати попередньо створено базу даних **PhoneDB.sdf** у проект **PhoneDiery**, викликавши контекстне меню по імені проекту у вікні Solution Explorer (Обозреватель решений) і обравши команду **Добавить (Add) – Существующий элемент (Existing item)** та вибравши за допомогою кнопки **Обзор базы даних PhoneDB.sdf**. Після цього база даних з'явиться у вікні проекту (рис. 2).

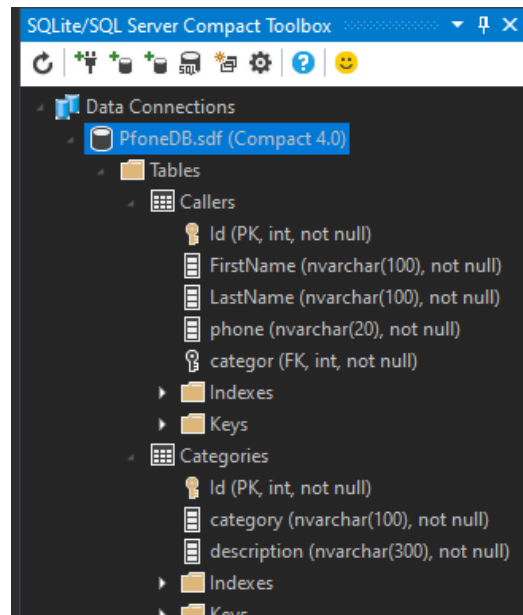


Рисунок 8 – Вигляд вікна проекту PhoneDiery із інтегрованою в нього БД PhoneDB.sdf

20. У вікні **Обозревателя решений (Solution Explorer)** додати файли з описами класів **Category** та **Caller**, призначених для роботи з відповідними таблицями бази даних PhoneDB.sdf, викликавши контекстне меню для **Заголовочные файлы (Header files)** і вибравши у ньому **Добавить (Add) – Создать элемент (New item) – Заголовочный файл (.h), Header file .h**, та ввівши спочатку ім'я **Category.h**, а після цього створивши файл **Caller.h**.

21. У відповідних файлах описати класи, призначені для опрацювання інформації згідно предметної області (ПО) Телефонний довідник. Таким чином, клас **Category**, описаний у файлі **Category.h**, буде мати наступний вигляд:

```
#pragma once
#include <stdexcept>
#include <String.h>
// підключення просторів імен std і System
using namespace std;
using namespace System;
// оголошення класу
ref class Category {
    //закриті поля класу для збереження даних про назву, опис та номер категорії
    String^ name;
    String^ describe;
    int id;
public: // відкриті члени класу
    Category() {} // порожній конструктор
    // конструктор ініціалізації полів об'єктів класу
    Category(String^ n, String^ d, int i) {
        this->name = n;
        this->describe = d;
```

```

        if (i > 0) {
            this->id = i;
        }
        else throw invalid_argument("Wrong id");
        // обробка помилки при спробі ввести код запису, менший або рівний 0
    }
    void setName(String^ category) {
        if (category->Length != 0) {
            this->name = category;
        }
        else throw invalid_argument("Wrong category");
        // обробка помилки при спробі залишити поле порожнім
    }
    String^ getName() {
        return this->name;
    }
    void setDescribe(String^ desc) {
        if (desc->Length != 0) {
            this->describe = desc;
        }
        else throw invalid_argument("Wrong describe");
        // обробка помилки при спробі залишити поле порожнім
    }
    String^ getDescribe() {
        return this->describe;
    }
    void setId(int id) {
        if (id > 0) {
            this->id = id;
        }
        else throw invalid_argument("Wrong id");
        // обробка помилки при спробі ввести код запису, менший або рівний 0
    }
    int getId() {
        return this->id;
    }
};

```

22. А в доданому файлі **Caller.h** клас, призначеному для опрацювання інформації про абонентів телефонного довідника, клас **Caller** буде мати наступний вигляд:

```

#pragma once
#include <stdexcept>
#include <String.h>
// підключення просторів імен std і System
using namespace std;
using namespace System;
// оголошення класу
ref class Caller {
    //закриті поля класу для збереження даних про ім'я, прізвище, номер телефону,
    номер //категорії контактів та номер контакту у записнику
    String^ name;
    String^ surname;
    String^ telNumber;
    int category;
    int id;
public: // відкриті члени класу

```

```
    Caller() {} // порожній конструктор
// конструктор ініціалізації полів об'єктів класу
    Caller(String^ n, String^ s, String^ t, int c, int i) {
        this->name = n;
        this->surname = s;
        this->telNumber = t;
        this->category = c;
        if (i > 0) {
            this->id = i;
        }
        else throw invalid_argument("Wrong id");
        // обробка помилки при спробі ввести код запису, менший або рівний 0
    }

    void setName(String^ firstName) {
        if (firstName->Length != 0) {
            this->name = firstName;
        }
        else throw invalid_argument("Wrong first name");
        // обробка помилки при спробі залишити поле порожнім
    }
    String^ getName() {
        return this->name;
    }
    void setSurname(String^ lastName) {
        if (lastName->Length != 0) {
            this->surname = lastName;
        }
        else throw invalid_argument("Wrong last name");
        // обробка помилки при спробі залишити поле порожнім
    }
    String^ getSurname() {
        return this->surname;
    }
    void setTelNumber(String^ telNumber) {
        if (telNumber->Length != 0) {
            this->telNumber = telNumber;
        }
        else throw invalid_argument("Wrong tel");
        // обробка помилки при спробі залишити поле порожнім
    }
    String^ getTelNumber() {
        return this->telNumber;
    }
    void setCategory(int category) {
        if (category > 0) {
            this->category = category;
        }
        else throw invalid_argument("Wrong category");
        // обробка помилки при спробі залишити поле порожнім
    }
    int getCategory() {
        return this->category;
    }
    String^ getFullName() {
        return this->name + " " + this->surname;
    }
    void setId(int id) {
        if (id > 0) {
            this->id = id;
        }
    }
}
```



```

    }
    else throw invalid_argument("Wrong id");
    // обробка помилки при спробі ввести код запису, менший рівний 0
}
int getId() {
    return this->id;
}
};

```

23. Для забезпечення роботи пункту головного меню **Категорії** слід додати форму для створення візуальної частини програми для роботи з даними таблиці **Categories** БД **PhoneDB.sdf**: у вікні **Обозреватель решений (Solution explorer)** обрати контекстне меню для назви проекту і у ньому вибрати **Добавить (Add) – Создать элемент (Add new item) – Форма Windows Form** та ввести ім'я файлу **Groups**. Створити графічний інтерфейс користувача згідно рисунку 9, розмістивши на формі компоненти **Label**, **TextBox**, **DataGridView** та **Button** і змінивши відповідні їх властивості до рисунку, наведеного нижче. Виконати пункт меню **Построение (Build) – Построить решение (Build solution)**.

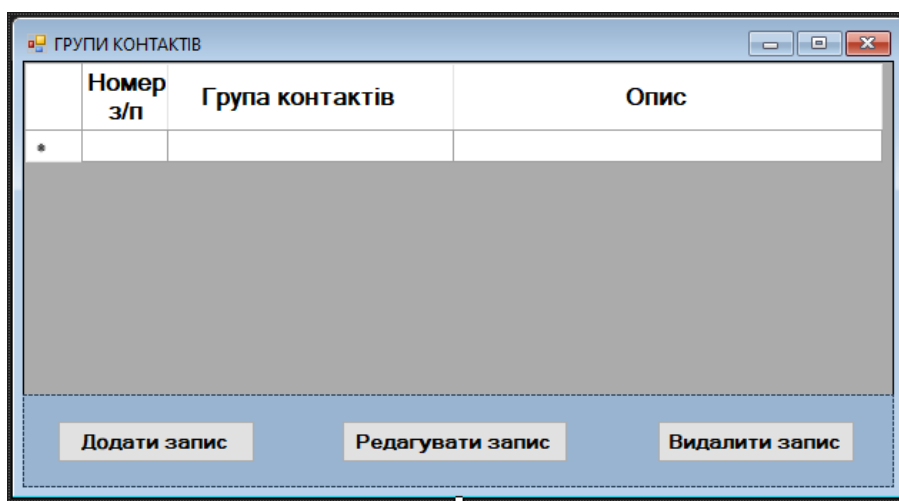


Рисунок 9 – Інтерфейс форми Groups, призначеної для роботи з категоріями контактів

24. Додати ще одну форму з графічним інтерфейсом для забезпечення введення даних про категорії контактів (рисунок 10), дати їй ім'я **EditGroups** та розмістити на ній компоненти **Label**, **TextBox** і **Button**, змінивши відповідні властивості цих компонентів до рисунку нижче.

Виконати пункт меню **Построение (Build) – Построить решение (Build solution)**.

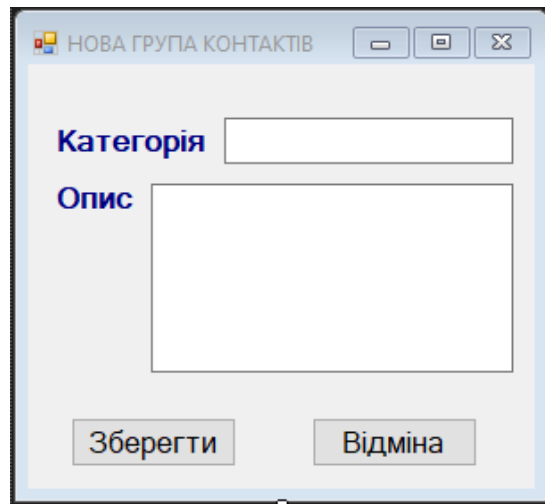


Рисунок 10 – Інтерфейс форми EditGroup для додавання/зміни даних про категорії контактів

25. Аналогічним чином створити графічний інтерфейс користувача (рисунок 11) для роботи з контактами, забезпечивши відображення на формі не тільки всіх контактів, а й контактів, категорія яких обрана користувачем, що забезпечується за рахунок реалізації зв'язку “один до багатьох” між таблицями categories та callers бази даних **PhoneDB.sdf**.

Для цього використати компоненти **Label**, **TextBox**, **DataGridView**, **Button**, **MenuStrip** з пунктом меню **Дані** (рисунок 11) і **ComboBox** (для створення випадаючого списку і вибору категорії контактів).

Виконати пункт меню **Построение (Build) – Построить решение (Build solution)**.

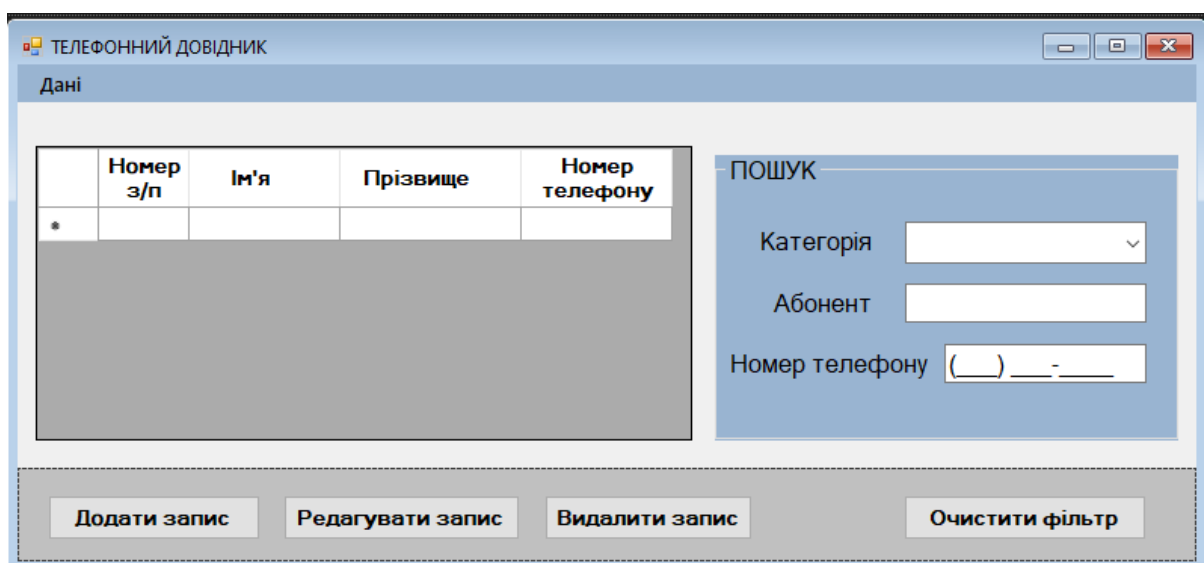


Рисунок 11 – Інтерфейс форми Kontakts для роботи з контактами

26. На рисунку, наведеному нижче, зображено вигляд пункту меню **Дані**, який містить команди **Сортувати за прізвищем**, **Сортувати за номером телефону**, **Сортувати за порядковим номером** і **Всі контакти**:

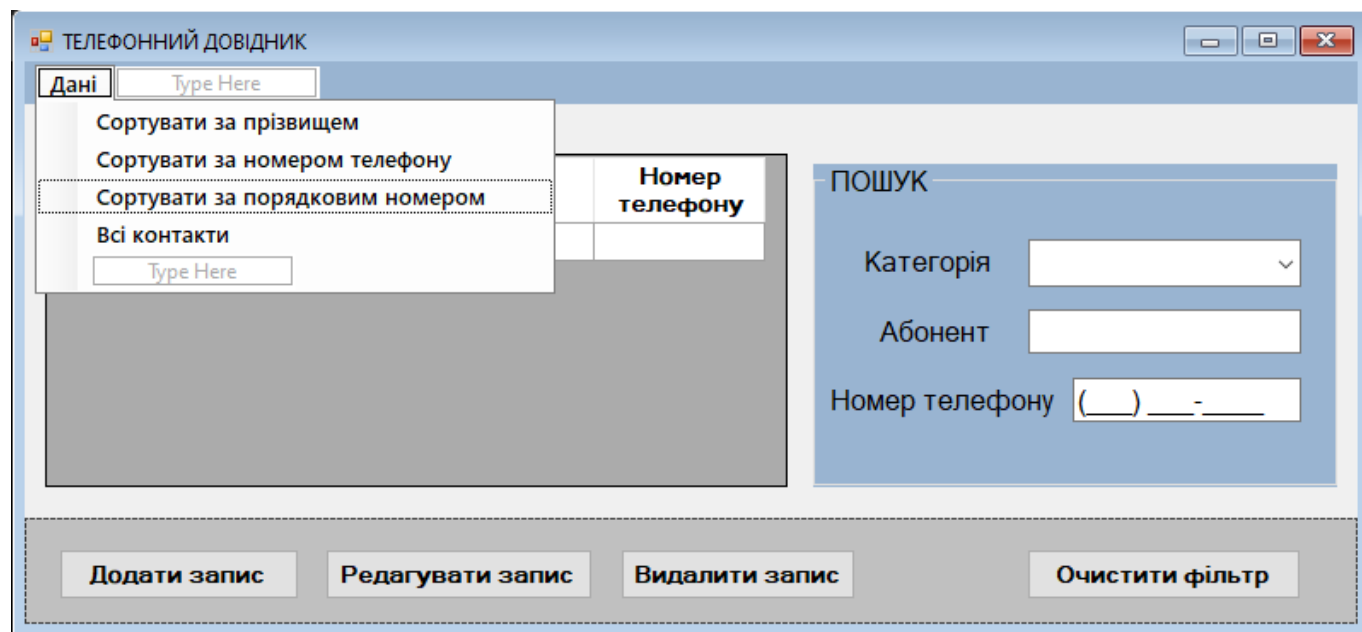


Рисунок 12 – Вигляд меню **Дані** форми **Kontakts** у режимі конструктора

27. Додати ще одну форму з графічним інтерфейсом для забезпечення введення даних про категорії контактів (рисунок 13), дати їй ім'я **EditKontakt** та розмістити на ній компоненти **Label**, **TextBox**, **maskedTextBox**, **comboBox** і **Button**, змінивши відповідні властивості цих компонентів до рисунку нижче. Виконати пункт меню **Построение (Build) – Побудувати рішення (Build solution)**.

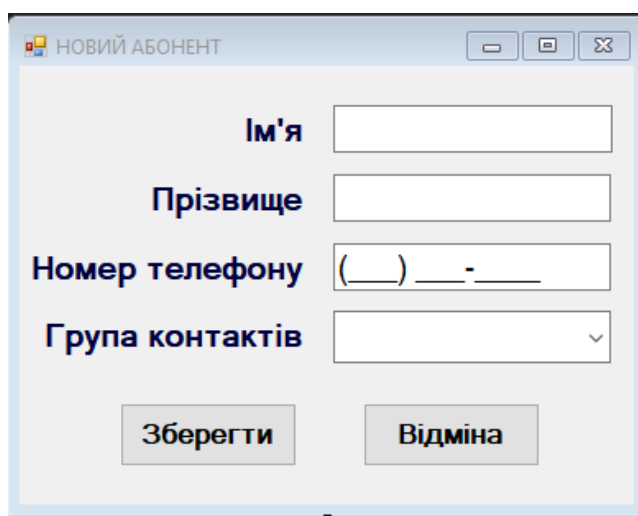


Рисунок 13 – Вікно форми введення/редагування даних про контакти у режимі конструктора

28. Перед тим, як почати писати програмну реалізацію для роботи з базою даних, слід підключити динамічну бібліотеку *System.Data.SqlServerCe*, додавши у проєкті посилання на неї.

29. Для цього потрібно викликати контекстне меню для імені проєкту у вікні **Обозреватель решений (Solution explorer)** та обрати з нього команду **Ссылки (References)**, так як це зображено на рисунку 14.

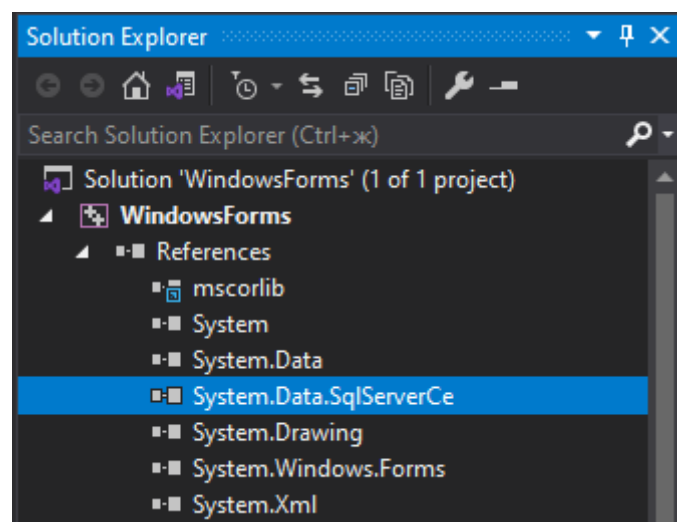
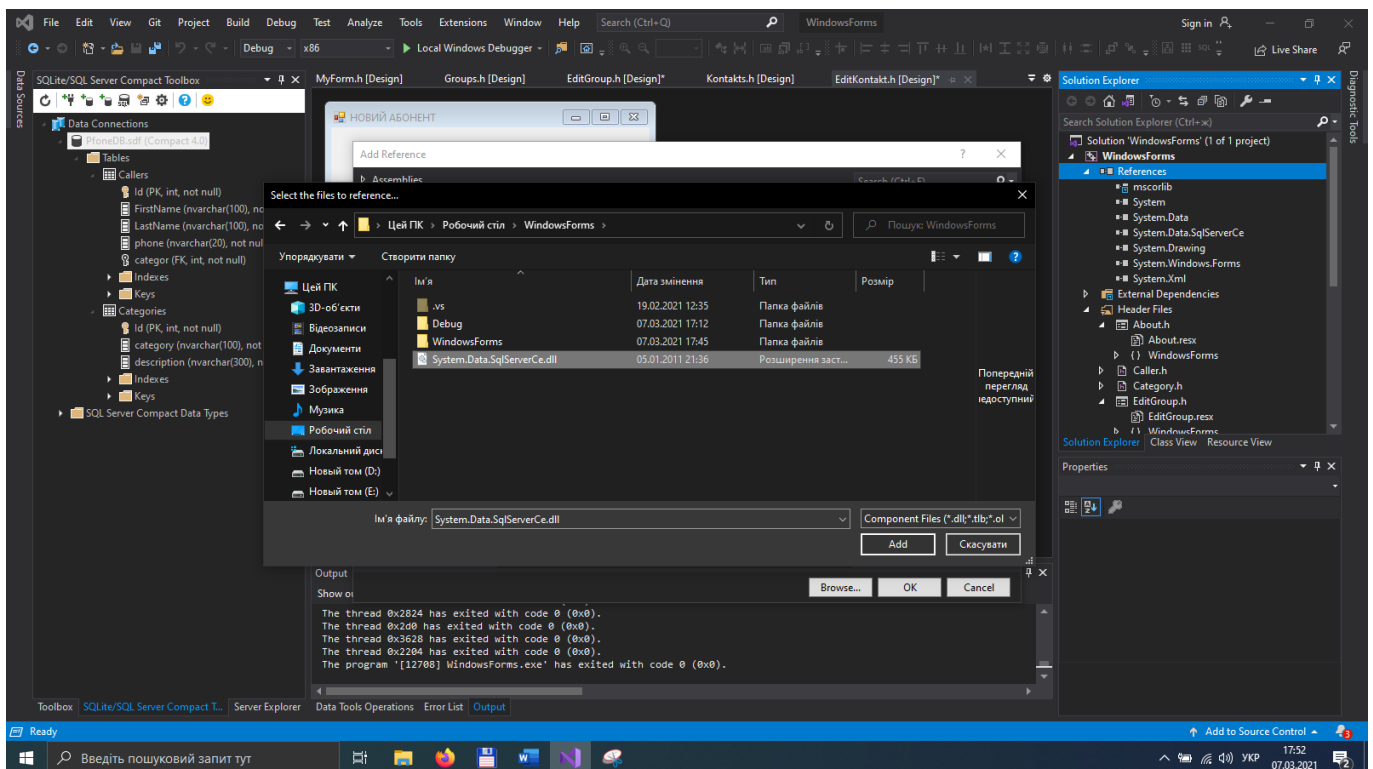


Рисунок 14 – Вигляд вікна пошуку бібліотеки *System.Data.SqlServerCe*

30. Перейти до форми **EditGroup.h** [Конструктор].

31. За допомогою контекстного меню, викликаного для форми, обрати команду **Перейти к коду**, після чого з'явиться заготовка файлу з кодом опису форми **EditGroup.h**.

32. У даній заготовці дописати програмну реалізацію для роботи з даними таблиці **categories** бази даних PhoneDB.sdf. Для цього спочатку слід підключити простір імен для роботи з sql-базою даних з розширенням .sdf, дописавши у програмний код, що відповідає за підключення просторів імен (рисунок 15),

`using namespace System::Data::SqlServerCe;`

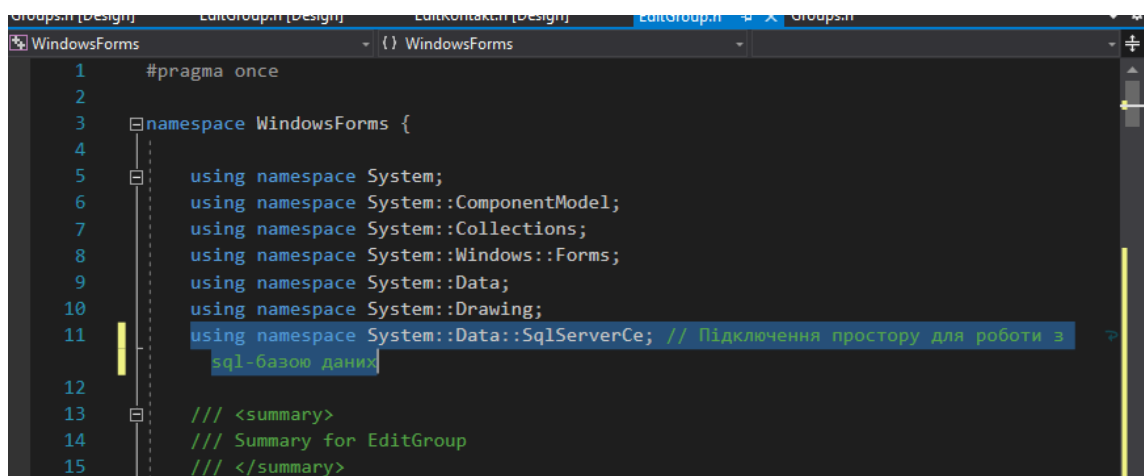


Рисунок 15 – Підключення простору імен System::Data::SqlServerCe

33. У описі класу форми **EditGroup** описати змінну **connect** для збереження рядка з'єднання з базою даних і змінну **categoryId** для запам'ятовування номера категорії (рисунок 16).

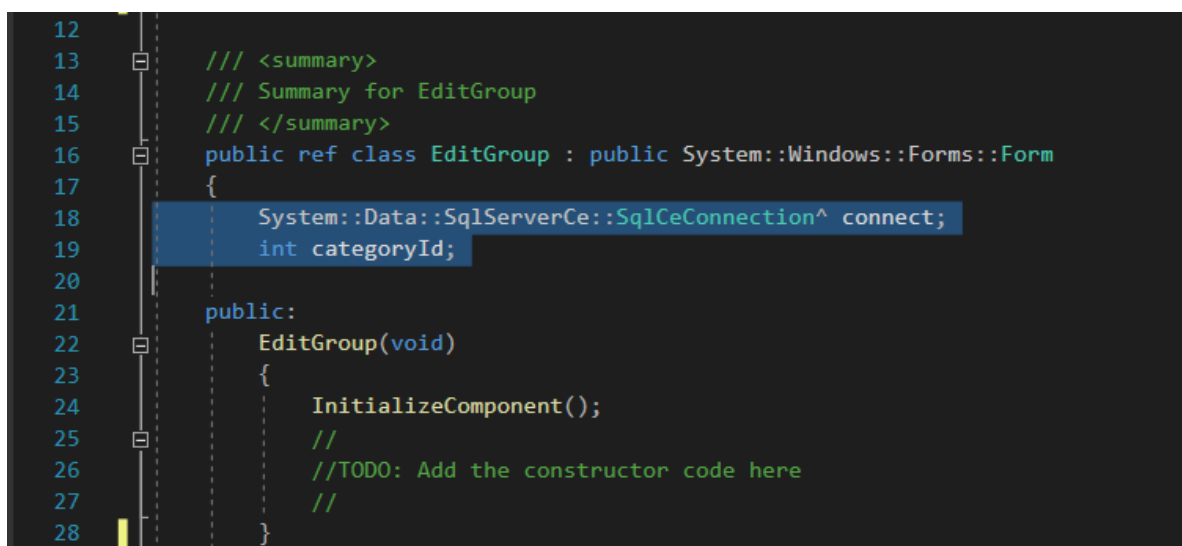
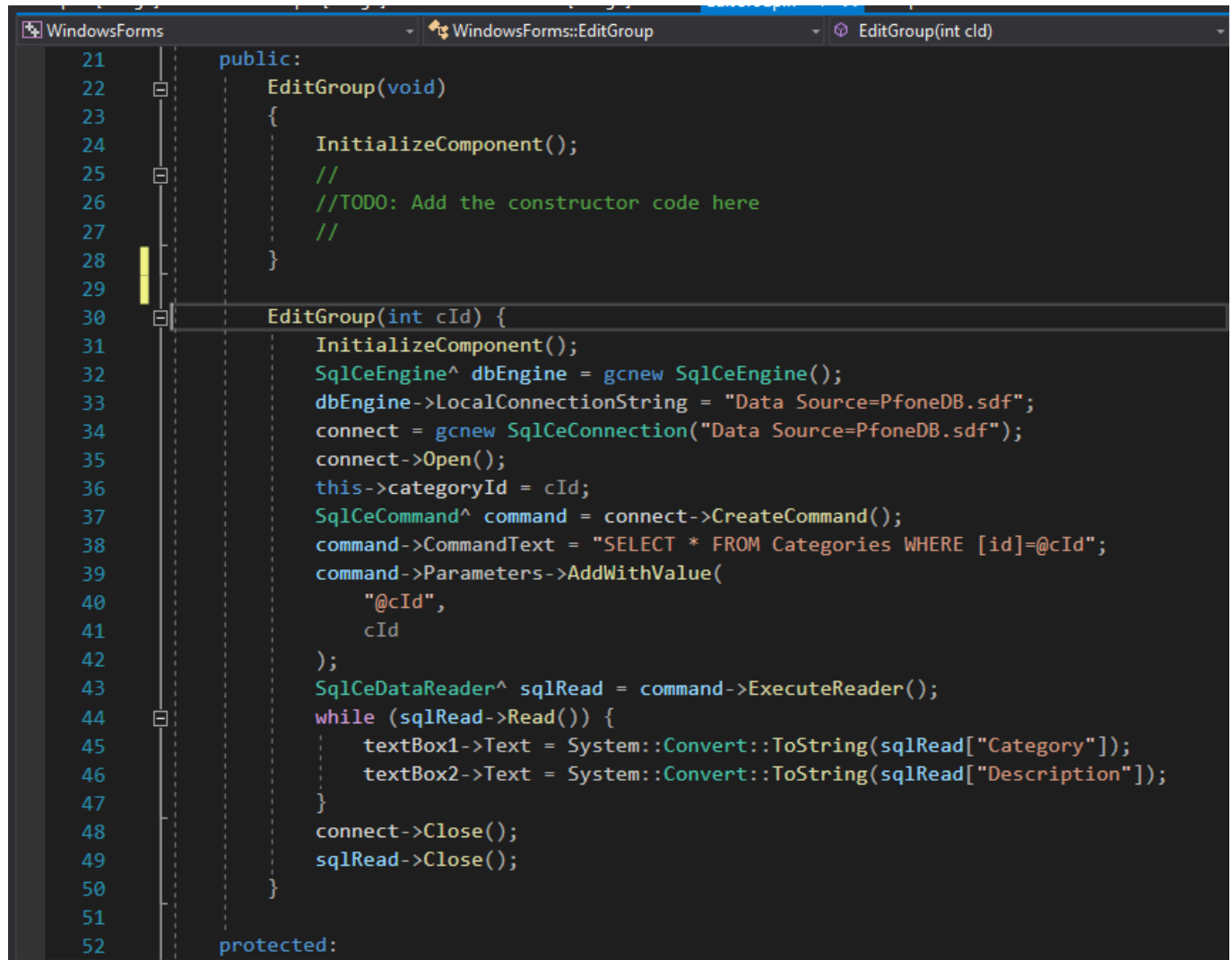


Рисунок 16 – Оголошення змінних класу EditGroup

34. Після порожнього конструктора форми **EditGroup (void)** визначити конструктор з параметрами **EditGroup (int cId)** для створення з'єднання з базою даних і зчитування даних про категорії контактів (рисунок. 17).



```
21 public:
22     EditGroup(void)
23     {
24         InitializeComponent();
25         //
26         //TODO: Add the constructor code here
27         //
28     }
29
30     EditGroup(int cId) {
31         InitializeComponent();
32         SqlCeEngine^ dbEngine = gcnew SqlCeEngine();
33         dbEngine->LocalConnectionString = "Data Source=PfoneDB.sdf";
34         connect = gcnew SqlCeConnection("Data Source=PfoneDB.sdf");
35         connect->Open();
36         this->categoryId = cId;
37         SqlCeCommand^ command = connect->CreateCommand();
38         command->CommandText = "SELECT * FROM Categories WHERE [id]=@cId";
39         command->Parameters->AddWithValue(
40             "@cId",
41             cId
42         );
43         SqlCeDataReader^ sqlRead = command->ExecuteReader();
44         while (sqlRead->Read()) {
45             textBox1->Text = System::Convert::ToString(sqlRead["Category"]);
46             textBox2->Text = System::Convert::ToString(sqlRead["Description"]);
47         }
48         connect->Close();
49         sqlRead->Close();
50     }
51
52     protected:
```

Рисунок 17 – Визначення конструктора з параметрами EditGroup (int cId)

35. Після розділу опису компонент, розміщених на формі, слід написати методи додавання та модифікації записів про категорії контактів (рисунок 18).


```

175     }
176     #pragma endregion
177
178     // Редагування запису
179     private: void update() {
180         connect->Open();
181         String^ query = "UPDATE Categories SET [category]='' + textBox1->Text +
182             '', [description]='' + textBox2->Text +
183             '' WHERE [id]='' + this->categoryId + "';";
184         SqlCeCommand^ command = connect->CreateCommand();
185         command->CommandText = query;
186         if (command->ExecuteNonQuery() == 0) {
187             MessageBox::Show("Updated!");
188         }
189         connect->Close();
190     }
191
192     // Додавання запису
193     private: void insert() {
194         connect->Open();
195         String^ query = "INSERT INTO Categories([category], [description])" +
196             "VALUES('' + textBox1->Text + "''', '' + textBox2->Text + "''')";
197         SqlCeCommand^ command = connect->CreateCommand();
198         command->CommandText = query;
199         if (command->ExecuteNonQuery() > 0) {
200             MessageBox::Show("Inserted!");
201         }
202         connect->Close();
203     }

```

Рисунок 18 – Визначення методів додавання та зміни записів про категорії контактів

36. Перейти назад у режим конструктора форми **EditGroup.h [Конструктор]** та запрограмувати кнопку **Зберегти**, двічі клікнувши по ній, після чого з'явиться заготовка процедури обробки даної кнопки

```

private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^ e)
{
},

```

у яку слід вписати відповідний код. Аналогічно запрограмувати кнопку **Відміна**. Вигляд програмної реалізації процедур обробки даних кнопок наведено на рисунку 19.

```

205 private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
206     if (textBox1->Text->Length != 0 && textBox2->Text->Length != 0) {
207         if (this->categoryId == -1) insert();
208         else update();
209         this->Close();
210     }
211     else {
212         MessageBox::Show("Add more data");
213     }
214 }
215 private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
216     this->Close();
217 }

```

Рисунок 19 – Програмна реалізація обробки кнопок Зберегти та Відміна

37. Таким чином, програмний код файлу EditGroup.h набуде наступного вигляду:

```

#pragma once

namespace WindowsForms {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::Data::SqlServerCe; // Підключення простору для роботи з
    sql-базою даних
    /// <summary>
    /// Summary for EditGroup
    /// </summary>
    public ref class EditGroup : public System::Windows::Forms::Form
    {
        System::Data::SqlServerCe::SqlCeConnection^ connect;
        int categoryId;
    public:
        EditGroup(void)
        {
            InitializeComponent();
            //
            //TODO: Add the constructor code here
            //
        }
        EditGroup(int cId) {
            InitializeComponent();
            SqlCeEngine^ dbEngine = gcnew SqlCeEngine();
            dbEngine->LocalConnectionString = "Data Source=PfoneDB.sdf";
            connect = gcnew SqlCeConnection("Data Source=PfoneDB.sdf");
            connect->Open();
            this->categoryId = cId;
            SqlCeCommand^ command = connect->CreateCommand();
            command->CommandText = "SELECT * FROM Categories WHERE [id]=@cId";
            command->Parameters->AddWithValue(
                "@cId",
                cId
            );
        }
    };
}

```

```

    );
    SqlConnection^ sqlRead = command->ExecuteReader();
    while (sqlRead->Read()) {
        textBox1->Text =
System::Convert::ToString(sqlRead["Category"]);
        textBox2->Text =
System::Convert::ToString(sqlRead["Description"]);
    }
    connect->Close();
    sqlRead->Close();
}
protected:
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    ~EditGroup()
    {
        if (components)
        {
            delete components;
        }
    }
private: System::Windows::Forms::Button^ button2;
protected:
private: System::Windows::Forms::Button^ button1;
private: System::Windows::Forms::TextBox^ textBox2;
private: System::Windows::Forms::TextBox^ textBox1;
private: System::Windows::Forms::Label^ label2;
private: System::Windows::Forms::Label^ label1;
private:
    /// <summary>
    /// Required designer variable.
    /// </summary>
    System::ComponentModel::Container ^components;
#pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
        this->button2 = (gcnew System::Windows::Forms::Button());
        this->button1 = (gcnew System::Windows::Forms::Button());
        this->textBox2 = (gcnew System::Windows::Forms::TextBox());
        this->textBox1 = (gcnew System::Windows::Forms::TextBox());
        this->label2 = (gcnew System::Windows::Forms::Label());
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->SuspendLayout();
        //
        // button2
        //
        this->button2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 12, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->button2->Location = System::Drawing::Point(159, 198);
        this->button2->Name = L"button2";
        this->button2->Size = System::Drawing::Size(93, 28);
        this->button2->TabIndex = 19;
        this->button2->Text = L"Відміна";

```

```
        this->button2->UseVisualStyleBackColor = true;
        this->button2->Click += gcnew System::EventHandler(this,
&EditGroup::button2_Click);
        //
        // button1
        //
        this->button1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 12, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->button1->Location = System::Drawing::Point(24, 198);
        this->button1->Name = L"button1";
        this->button1->Size = System::Drawing::Size(93, 28);
        this->button1->TabIndex = 18;
        this->button1->Text = L"Збергти";
        this->button1->UseVisualStyleBackColor = true;
        this->button1->Click += gcnew System::EventHandler(this,
&EditGroup::button1_Click);
        //
        // textBox2
        //
        this->textBox2->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 12, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->textBox2->Location = System::Drawing::Point(69, 67);
        this->textBox2->Multiline = true;
        this->textBox2->Name = L"textBox2";
        this->textBox2->Size = System::Drawing::Size(203, 106);
        this->textBox2->TabIndex = 17;
        //
        // textBox1
        //
        this->textBox1->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 12, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->textBox1->Location = System::Drawing::Point(110, 30);
        this->textBox1->Name = L"textBox1";
        this->textBox1->Size = System::Drawing::Size(162, 26);
        this->textBox1->TabIndex = 16;
        //
        // label2
        //
        this->label2->AutoSize = true;
        this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 12, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->label2->ForeColor = System::Drawing::Color::Navy;
        this->label2->Location = System::Drawing::Point(12, 65);
        this->label2->Name = L"label2";
        this->label2->Size = System::Drawing::Size(51, 20);
        this->label2->TabIndex = 15;
        this->label2->Text = L"Опис";
        //
        // label1
        //
        this->label1->AutoSize = true;
        this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 12, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
```

```

        static_cast<System::Byte>(204));
this->label1->ForeColor = System::Drawing::Color::Navy;
this->label1->Location = System::Drawing::Point(12, 33);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(92, 20);
this->label1->TabIndex = 14;
this->label1->Text = L"Категорія";
//
// EditGroup
//
this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(284, 238);
this->Controls->Add(this->button2);
this->Controls->Add(this->button1);
this->Controls->Add(this->textBox2);
this->Controls->Add(this->textBox1);
this->Controls->Add(this->label2);
this->Controls->Add(this->label1);
this->Name = L"EditGroup";
this->Text = L"НОВА ГРУПА КОНТАКТІВ";
this->ResumeLayout(false);
this->PerformLayout();
}
#pragma endregion

// Редагування запису
private: void update() {
    connect->Open();
    String^ query = "UPDATE Categories SET [category]='" + textBox1-
>Text +
        "', [description]='" + textBox2->Text +
        "' WHERE [id]='" + this->categoryId + "';";
    SqlCeCommand^ command = connect->CreateCommand();
    command->CommandText = query;
    if (command->ExecuteNonQuery() == 0) {
        MessageBox::Show("Updated!");
    }
    connect->Close();
}

// Додавання запису
private: void insert() {
    connect->Open();
    String^ query = "INSERT INTO Categories([category], [description])" +
        "VALUES('" + textBox1->Text + "', '" + textBox2->Text + "');"
    SqlCeCommand^ command = connect->CreateCommand();
    command->CommandText = query;
    if (command->ExecuteNonQuery() > 0) {
        MessageBox::Show("Inserted!");
    }
    connect->Close();
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
    if (textBox1->Text->Length != 0 && textBox2->Text->Length != 0) {
        if (this->categoryId == -1) insert();
        else update();
        this->Close();
    }
    else {
        MessageBox::Show("Add more data");
    }
}

```

```

    }
}
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
    this->Close();
}
};
}

```

38. Перейти до коду для форми **Groups.h** і у заготовці, що з'явилась, спочатку коду підключити заголовочні файли **EditGroup.h** та **Category.h**, що містять описи відповідних класів для змоги використання їх, бібліотеки **stdlib.h** та просторів імен **System::Collections::Generic** для роботи з колекціями і **System::Data::SqlServerCe** для роботи з sdf-базою (рисунок 20):

```

2 // підключення заголовочних файлів
3 #include "EditGroup.h"
4 #include "Category.h"
5 #include "stdlib.h"
6
7 namespace WindowsForms {
8
9     using namespace System;
10    using namespace System::ComponentModel;
11    using namespace System::Collections;
12    using namespace System::Windows::Forms;
13    using namespace System::Data;
14    using namespace System::Drawing;
15    using namespace System::Collections::Generic;
16    using namespace System::Data::SqlServerCe; // Підключення простору для роботи з
17    sql-базою даних

```

Рисунок 20 – Підключення заголовочних файлів і просторів імен

39. У тілі опису класу **Groups** оголосити змінні для вміщення списку категорій контактів (**group**) і рядка з'єднання з БД (**connect**) згідно рисунку 21:

```

21 public ref class Groups : public System::Windows::Forms::Form
22 {
23     private: List<Category^>^ group;
24             System::Data::SqlServerCe::SqlCeConnection^ connect;
25     public:
26         Groups(void)
27         {
28             InitializeComponent();

```

Рисунок 21 – Оголошення змінних класу Groups

40. У тілі конструктора **Groups(void)** реалізувати створення з'єднання з базою даних і списку категорії контактів (рисунок 22):


```

25      public:
26      Groups(void)
27      {
28          InitializeComponent();
29          //
30          //TODO: Add the constructor code here
31          //
32          SqlCeEngine^ dbEngine = gcnew SqlCeEngine();
33          dbEngine->LocalConnectionString = "Data Source=PfoneDB.sdf";
34          connect = gcnew SqlCeConnection(dbEngine->LocalConnectionString);
35          group = gcnew List<Category^>();
36      }

```

Рисунок 22 – Конструктор без параметрів класу Groups

41. Нижче написати код методів зчитування даних про категорії контактів з БД (**getCategories()**) згідно рисунку 23:

```

37      void getCategories() {
38          connect->Open();
39          SqlCeCommand^ command = connect->CreateCommand();
40          command->CommandText = "SELECT * FROM Categories";
41          SqlCeDataReader^ sqlRead = command->ExecuteReader();
42          group->Clear();
43          while (sqlRead->Read()) {
44              group->Add(gcnew Category(
45                  System::Convert::ToString(sqlRead["category"]),
46                  System::Convert::ToString(sqlRead["description"]),
47                  System::Convert::ToInt32(sqlRead["id"])
48              ));
49          }
50          connect->Close();
51          this->updateTable();
52      }

```

Рисунок 23 – Метод читання даних про категорії контактів з БД

42. Написати код методу виконання запиту до БД (**execute()**) згідно рисунку 24:

```

53      void execute(String^ query) {
54          connect->Open();
55          SqlCeCommand^ command = gcnew SqlCeCommand(query, connect);
56          command->ExecuteNonQuery();
57          connect->Close();
58      }

```

Рисунок 24 – Метод виконання запиту до БД

43. Далі нижче визначити методи відображення даних про категорії контактів (**getQueryCategories()**) та оновлення записів у таблиці categories (**updateTable()**) згідно рисунку 25:

```

59 void getQueryCategories(String^ query) {
60     connect->Open();
61     SqlCeCommand^ command = gcnew SqlCeCommand(query, connect);
62     SqlCeDataReader^ sqlRead = command->ExecuteReader();
63     group->Clear();
64     while (sqlRead->Read()) {
65         group->Add(gcnew Category(
66             System::Convert::ToString(sqlRead["category"]),
67             System::Convert::ToString(sqlRead["description"]),
68             System::Convert::ToInt32(sqlRead["id"])
69         ));
70     }
71     sqlRead->Close();
72     connect->Close();
73     this->updateTable();
74 }
75 //
76 void updateTable() {
77     int i = 0;
78     dataGridView1->Rows->Clear();
79     for each (Category ^ category in group) {
80         this->dataGridView1->Rows->Add();
81         this->dataGridView1->Rows[i]->Cells[0]->Value = category->getId();
82         this->dataGridView1->Rows[i]->Cells[1]->Value = category->getName();
83         this->dataGridView1->Rows[i]->Cells[2]->Value = category->getDescribe
84             ();
85         ++i;
86     }
87 }

```

Рисунок 25 – Методи виведення даних про категорії контактів на форму та оновлення записів відповідної таблиці

44. Запрограмувати кнопки **Додати запис**, **Редагувати запис** та **Видалити запис**, двічі клікаючи по цих кнопках та вписуючи відповідну їхню програмну реалізацію, і процедуру обробки завантаження форми EditGroup за допомогою подвійного кліку по ній:

```

//Запрограмування відображення даних при запуску форми
private: System::Void Groups_Load(System::Object^ sender, System::EventArgs^ e) {
    this->getCategories();
}

// Запрограмування кнопки Додати запис
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
    EditGroup^ edit = gcnew EditGroup(-1);
    edit->ShowDialog();
    this->getCategories();
}

// Запрограмування кнопки Редагувати запис

```

```
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
    EditGroup^ edit = gcnew EditGroup(System::Convert::ToInt32(dataGridView1-
>Rows[dataGridView1->CurrentCell->RowIndex]->Cells[0]->Value));
    edit->ShowDialog();
    this->getCategories();
}

// Запрограмування кнопки Видалити запис
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
    try {
        connect->Open();
        String^ query = "DELETE FROM Categories WHERE [id]=" +
System::Convert::ToString(dataGridView1->Rows[dataGridView1->CurrentCell->RowIndex]-
>Cells[0]->Value);
        SqlCeCommand^ command = gcnew SqlCeCommand(query, connect);
        if (command->ExecuteNonQuery() > 0) {
            MessageBox::Show("Deleted!");
        }
        connect->Close();
        this->getCategories();
    }
    catch (...) {
        MessageBox::Show("Choose correct item");
    }
}
```

Таким чином, повний програмний код файлу **Groups.h** наведено нижче:

```
#pragma once
// підключення заголовочних файлів
#include "EditGroup.h"
#include "Category.h"
#include "stdlib.h"

namespace WindowsForms {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::Collections::Generic;
    using namespace System::Data::SqlServerCe; // Підключення простору для роботи з sql-базою
даних

    /// <summary>
    /// Summary for Groups
    /// </summary>
    public ref class Groups : public System::Windows::Forms::Form
    {
    private: List<Category^>^ group;
        System::Data::SqlServerCe::SqlCeConnection^ connect;
    public:
        Groups(void)
        {
            InitializeComponent();
            //
            //TODO: Add the constructor code here
            //
            SqlCeEngine^ dbEngine = gcnew SqlCeEngine();
        }
    };
}
```

```

        dbEngine->LocalConnectionString = "Data Source=PfoneDB.sdf";
        connect = gcnew SqlConnection(dbEngine->LocalConnectionString);
        group = gcnew List<Category^>();
    }
        SqlCeEngine^ dbEngine = gcnew SqlCeEngine();
        dbEngine->LocalConnectionString = "Data Source=\"PhoneDB.sdf\"";
        connect = gcnew SqlConnection(dbEngine->LocalConnectionString);
        group=gcnew List<Category^>();
    }
// метод відображення на формі даних про категорії контактів
void getCategories() {
    connect->Open();
    SqlCeCommand^ command = connect->CreateCommand();
    command->CommandText = "SELECT * FROM Categories";
    SqlCeDataReader^ sqlRead = command->ExecuteReader();
    group->Clear();
    while (sqlRead->Read()) {
        group->Add(gcnew Category(
            System::Convert::ToString(sqlRead["category"]),
            System::Convert::ToString(sqlRead["description"]),
            System::Convert::ToInt32(sqlRead["id"])
        ));
    }
    connect->Close();
    this->updateTable();
}
//метод передачі запиту до БД на виконання
void execute(String^ query) {
    connect->Open();
    SqlCeCommand^ command = gcnew SqlCeCommand(query, connect);
    command->ExecuteNonQuery();
    connect->Close();
}
// метод повернення результатів виконання запиту
void getQueryCategories(String^ query) {
    connect->Open();
    SqlCeCommand^ command = gcnew SqlCeCommand(query, connect);
    SqlCeDataReader^ sqlRead = command->ExecuteReader();
    group->Clear();
    while (sqlRead->Read()) {
        group->Add(gcnew Category(
            System::Convert::ToString(sqlRead["category"]),
            System::Convert::ToString(sqlRead["description"]),
            System::Convert::ToInt32(sqlRead["id"])
        ));
    }
    sqlRead->Close();
    connect->Close();
    this->updateTable();
}
// метод оновлення таблиці dataGridView
void updateTable() {
    int i = 0;
    dataGridView1->Rows->Clear();
    for each (Category ^ category in group) {
        this->dataGridView1->Rows->Add();
        this->dataGridView1->Rows[i]->Cells[0]->Value = category->getId();
        this->dataGridView1->Rows[i]->Cells[1]->Value = category->getName();
        this->dataGridView1->Rows[i]->Cells[2]->Value = category->getDescribe();
        ++i;
    }
}
}

```

```

protected:
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    ~Groups()
    {
        if (components)
        {
            delete components;
        }
    }
private: System::Windows::Forms::Button^ button3;
protected:
private: System::Windows::Forms::Button^ button2;
private: System::Windows::Forms::Button^ button1;
private: System::Windows::Forms::DataGridView^ dataGridView1;
private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column1;
private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column2;
private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column3;
private: System::Windows::Forms::Panel^ panel1;

private:
    /// <summary>
    /// Required designer variable.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
        System::Windows::Forms::DataGridViewCellStyle^ dataGridViewCellStyle1 =
(gcnew System::Windows::Forms::DataGridViewCellStyle());
        this->button3 = (gcnew System::Windows::Forms::Button());
        this->button2 = (gcnew System::Windows::Forms::Button());
        this->button1 = (gcnew System::Windows::Forms::Button());
        this->dataGridView1 = (gcnew System::Windows::Forms::DataGridView());
        this->Column1 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());
        this->Column2 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());
        this->Column3 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());
        this->panel1 = (gcnew System::Windows::Forms::Panel());
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>dataGridView1))->BeginInit();
        this->panel1->SuspendLayout();
        this->SuspendLayout();
        //
        // button3
        //
        this->button3->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif",
9.75F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
        this->button3->Location = System::Drawing::Point(444, 18);
        this->button3->Name = L"button3";
        this->button3->Size = System::Drawing::Size(139, 30);
        this->button3->TabIndex = 20;
        this->button3->Text = L"Видалити запис";
        this->button3->UseVisualStyleBackColor = true;
    }

```

```

        this->button3->Click += gcnew System::EventHandler(this,
&Groups::button3_Click);
        //
        // button2
        //
        this->button2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif",
9.75F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->button2->Location = System::Drawing::Point(223, 18);
        this->button2->Name = L"button2";
        this->button2->Size = System::Drawing::Size(159, 30);
        this->button2->TabIndex = 19;
        this->button2->Text = L"Редагувати запис";
        this->button2->UseVisualStyleBackColor = true;
        this->button2->Click += gcnew System::EventHandler(this,
&Groups::button2_Click);
        //
        // button1
        //
        this->button1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif",
9.75F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->button1->Location = System::Drawing::Point(24, 18);
        this->button1->Name = L"button1";
        this->button1->Size = System::Drawing::Size(139, 30);
        this->button1->TabIndex = 18;
        this->button1->Text = L"Додати запис";
        this->button1->UseVisualStyleBackColor = true;
        this->button1->Click += gcnew System::EventHandler(this,
&Groups::button1_Click);
        //
        // dataGridView1
        //
        dataGridViewCellStyle1->Alignment =
System::Windows::Forms::DataGridViewContentAlignment::MiddleCenter;
        dataGridViewCellStyle1->BackColor = System::Drawing::SystemColors::Control;
        dataGridViewCellStyle1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 12, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        dataGridViewCellStyle1->ForeColor =
System::Drawing::SystemColors::WindowText;
        dataGridViewCellStyle1->SelectionBackColor =
System::Drawing::SystemColors::Highlight;
        dataGridViewCellStyle1->SelectionForeColor =
System::Drawing::SystemColors::HighlightText;
        dataGridViewCellStyle1->WrapMode =
System::Windows::Forms::DataGridViewTriState::True;
        this->dataGridView1->ColumnHeadersDefaultCellStyle = dataGridViewCellStyle1;
        this->dataGridView1->ColumnHeadersHeightSizeMode =
System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;
        this->dataGridView1->Columns->AddRange(gcnew cli::array<
System::Windows::Forms::DataGridViewColumn^ >(3) {
            this->Column1,
            this->Column2, this->Column3
        });
        this->dataGridView1->Dock = System::Windows::Forms::DockStyle::Fill;
        this->dataGridView1->Location = System::Drawing::Point(0, 0);
        this->dataGridView1->Name = L"dataGridView1";
        this->dataGridView1->Size = System::Drawing::Size(610, 298);
        this->dataGridView1->TabIndex = 17;
        //

```



```

        // Column1
        //
        this->Column1->HeaderText = L"Номер з/п";
        this->Column1->Name = L"Column1";
        this->Column1->Width = 60;
        //
        // Column2
        //
        this->Column2->HeaderText = L"Група контактів";
        this->Column2->Name = L"Column2";
        this->Column2->Width = 200;
        //
        // Column3
        //
        this->Column3->HeaderText = L"Опис";
        this->Column3->Name = L"Column3";
        this->Column3->Width = 300;
        //
        // panel1
        //
        this->panel1->BackColor = System::Drawing::SystemColors::ActiveCaption;
        this->panel1->Controls->Add(this->button3);
        this->panel1->Controls->Add(this->button1);
        this->panel1->Controls->Add(this->button2);
        this->panel1->Dock = System::Windows::Forms::DockStyle::Bottom;
        this->panel1->Location = System::Drawing::Point(0, 233);
        this->panel1->Name = L"panel1";
        this->panel1->Size = System::Drawing::Size(610, 65);
        this->panel1->TabIndex = 21;
        //
        // Groups
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
        this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
        this->ClientSize = System::Drawing::Size(610, 298);
        this->Controls->Add(this->panel1);
        this->Controls->Add(this->dataGridView1);
        this->Name = L"Groups";
        this->Text = L"ГРУПИ КОНТАКТІВ";
        this->Load += gcnew System::EventHandler(this, &Groups::Groups_Load);
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>dataGridView1))->EndInit();
        this->panel1->ResumeLayout(false);
        this->ResumeLayout(false);
    }
#pragma endregion
//Запрограмування відображення даних при запуску форми
private: System::Void Groups_Load(System::Object^ sender, System::EventArgs^ e) {
    this->getCategories();
}
// Запрограмування кнопки Додати запис
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
    EditGroup^ edit = gcnew EditGroup(-1);
    edit->ShowDialog();
    this->getCategories();
}
// Запрограмування кнопки Редагувати запис
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
    EditGroup^ edit = gcnew EditGroup(System::Convert::ToInt32(dataGridView1-
>Rows[dataGridView1->CurrentCell->RowIndex]->Cells[0]->Value));

```

```

        edit->ShowDialog();
        this->getCategories();
    }

    // Запрограмування кнопки Видалити запис
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
    try {
        connect->Open();
        String^ query = "DELETE FROM Categories WHERE [id]=" +
System::Convert::ToString(dataGridView1->Rows[dataGridView1->CurrentCell->RowIndex]->Cells[0]-
>Value);

        SqlCeCommand^ command = gcnew SqlCeCommand(query, connect);
        if (command->ExecuteNonQuery() > 0) {
            MessageBox::Show("Deleted!");
        }
        connect->Close();
        this->getCategories();
    }
    catch (...) {
        MessageBox::Show("Choose correct item");
    }
}
};
}

```

45. Перейти до визначення програмної реалізації форми Edit.h.

46. Підключити простір імен для роботи з sdf-файлом БД:

```

#pragma once

namespace WindowsForms {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::Data::SqlServerCe; // Підключення простору для роботи з sql-базою
даних
}

```

47. Оголосити закриті змінні класу **Edit**: **connect** — для збереження рядка з'єднання з БД, **personId** — для вміщення номеру контакту, а **groupId** — для запам'ятовування номеру категорії контактів:

```

/// <summary>
/// Summary for EditKontakt
/// </summary>
public ref class EditKontakt : public System::Windows::Forms::Form
{
private:
    System::Data::SqlServerCe::SqlCeConnection^ connect;
    int personId;
    int groupId;
}

```

48. Визначити конструктори з параметрами для створення з'єднання з БД та відображення даних про контакти обраної категорії:

```
public:
    EditKontakt(int temp, int groupid)
    {
        InitializeComponent();
        //
        //TODO: Add the constructor code here
        //

        SqlCeEngine^ dbEngine = gcnew SqlCeEngine();
        dbEngine->LocalConnectionString = "Data Source=PfoneDB.sdf;";
        connect = gcnew SqlCeConnection("Data Source=PfoneDB.sdf;");
        this->personId = -1;
        this->groupId = groupid;
        textBox1->Text = "";
        textBox2->Text = "";

        connect->Open();
        SqlCeCommand^ command = connect->CreateCommand();
        command->CommandText = "SELECT category FROM Categories";
        SqlCeDataReader^ sqlRead = command->ExecuteReader();
        while (sqlRead->Read()) {
            comboBox1->Items->Add(System::Convert::ToString(sqlRead["category"]));
        }
        connect->Close();
        sqlRead->Close();
    }

    EditKontakt(int pId) {
        InitializeComponent();
        SqlCeEngine^ dbEngine = gcnew SqlCeEngine();
        dbEngine->LocalConnectionString = "Data Source=PfoneDB.sdf;";
        connect = gcnew SqlCeConnection("Data Source=PfoneDB.sdf;");
        connect->Open();
        SqlCeCommand^ command = connect->CreateCommand();
        command->CommandText = "SELECT category FROM Categories";
        SqlCeDataReader^ sqlRead = command->ExecuteReader();
        while (sqlRead->Read()) {
            comboBox1->Items->Add(System::Convert::ToString(sqlRead["category"]));
        }
        connect->Close();
        sqlRead->Close();
        this->personId = pId;
        int cCB;
        connect->Open();
        SqlCeCommand^ command2 = connect->CreateCommand();
        command2->CommandText = "SELECT * FROM Callers WHERE [id]=@pId";
        command2->Parameters->AddWithValue("@pId", pId);
        SqlCeDataReader^ sqlRead2 = command2->ExecuteReader();
        while (sqlRead2->Read()) {
            textBox1->Text = System::Convert::ToString(sqlRead2["FirstName"]);
            textBox2->Text = System::Convert::ToString(sqlRead2["LastName"]);
            maskedTextBox1->Text = System::Convert::ToString(sqlRead2["phone"]);
            cCB = System::Convert::ToInt32(sqlRead2["categor"]);
        }
        connect->Close();
        sqlRead2->Close();
        this->currComboBoxSet(cCB);
    }
}
```

49. Написати методи **currComboBoxSet** і **getCurrComboBox** для зчитування і занесення даних в **comboBox**

```
void currComboBoxSet(int id)
{
    connect->Open();
    String^ query = "SELECT category FROM Categories WHERE id LIKE '" + id + "'";
    SqlCeCommand^ command = connect->CreateCommand();
    command->CommandText = query;
    SqlCeDataReader^ sqlRead = command->ExecuteReader();
    bool hasRow = sqlRead->Read();
    if (hasRow) {
        comboBox1->Text = System::Convert::ToString(sqlRead["category"]);
    }
    connect->Close();
    sqlRead->Close();
}

int getCurrComboBox()
{
    int id;
    connect->Open();
    String^ queryS = "SELECT id FROM Categories WHERE category LIKE '" + comboBox1->Text + "'";
    SqlCeCommand^ commandS = connect->CreateCommand();
    commandS->CommandText = queryS;
    SqlCeDataReader^ sqlReadS = commandS->ExecuteReader();
    bool hasRow = sqlReadS->Read();
    if (hasRow) {
        id = System::Convert::ToInt32(sqlReadS["id"]);
    }
    sqlReadS->Close();
    connect->Close();
    return id;
}
```

50. Після опису компонент форми EditKontakt визначити програмну реалізацію методів зміни та додавання записів таблиці Callers відповідно:

```
#pragma endregion

private: void update() {
    int cID = this->getCurrComboBox();
    connect->Open();
    String^ query = "UPDATE Callers SET [FirstName]='" + textBox1->Text +
        "', [LastName]='" + textBox2->Text +
        "', [phone]='" + maskedTextBox1->Text +
        "', [categor]='" + cID +
        " WHERE [id]='" + this->personId + "'";
    SqlCeCommand^ command = connect->CreateCommand();
    command->CommandText = query;
    if (command->ExecuteNonQuery() == 0) {
        MessageBox::Show("Оновлено!");
    }
    connect->Close();
}
```

```

private: void insert() {
    int cCB = this->getCurrComboBox();
    connect->Open();
    String^ query = "INSERT INTO Callers([FirstName], [LastName], [phone], [categor])" +
        "VALUES(' + textBox1->Text + "','" + textBox2->Text + "','" +
maskedTextBox1->Text + "','" + cCB + "');"
    SqlCeCommand^ command = connect->CreateCommand();
    command->CommandText = query;
    if (command->ExecuteNonQuery() > 0) {
        MessageBox::Show("Додано!");
    }
    else MessageBox::Show("Введіть усі данні!");
    connect->Close();
}

```

51. запрограмувати кнопки **Зберегти** та **Відміна** за допомогою подвійного кліку по них та програмної реалізації, наведеної нижче, відповідно:

```

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
    if (textBox1->Text->Length != 0 && textBox2->Text->Length != 0 ) {
        if (this->personId == -1) insert();
        else update();
        this->Close();
    }
    else {
        MessageBox::Show("Введіть більше даних");
    }
}
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
    Close();
}

```

52. Перейти до програмної реалізації форми **Kontakts**.

53. Підключити заголовочні файли з описом класів **Caller** та форми **Edit** і простори імен для роботи з sdf-файлом БД і колекціями об'єктів:

```

#pragma once
#include "Caller.h"
#include "EditKontakt.h"

namespace WindowsForms {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::Data::SqlServerCe; // Підключення простору для роботи з
sql-базою даних
    using namespace System::Collections::Generic;
}

```

54. Оголосити закриті змінні класу: **group** – для вміщення списку об'єктів з інформацією про контакти, **connect** – для збереження рядка з'єднання з БД, **groupId** – для запам'ятовування номера категорії контактів:

```
public ref class Kontakts : public System::Windows::Forms::Form
{
private: List<Caller^>^ group;
        System::Data::SqlServerCe::SqlCeConnection^ connect;
        int groupId;
```

55. Визначити конструктор без параметрів, забезпечивши в ньому створення з'єднання з БД та списку об'єктів для збереження інформації про контакти:

```
public:
    Kontakts(void)
    {
        InitializeComponent();
        //
        //TODO: Add the constructor code here
        //
        SqlCeEngine^ dbEngine = gcnew SqlCeEngine();
        dbEngine->LocalConnectionString = "Data Source=PfoneDB.sdf;";
        connect = gcnew SqlCeConnection(dbEngine->LocalConnectionString);
        group = gcnew List<Caller^>();
    }
```

56. Нижче конструктора написати методи відображення записів з контактами на формі (**getCallers()**), виконання запиту до БД (**execute()**), вибірки номеру категорії (**getSelectedGroupIndex()**), читання даних з таблиці callers (**getQueryCallers()**), відображення назви категорії (**getNameCategor()**) та оновлення даних у таблиці Callers (**updateTable()**) відповідно:

```
void getCallers()
{
    this->groupId = getSelectedGroupIndex();
    if (this->groupId < 0)
    {
        connect->Open();
        SqlCeCommand^ command = connect->CreateCommand();
        command->CommandText = "SELECT * FROM Callers";
        command->Parameters->AddWithValue(
            "@group",
            this->groupId
        );
        SqlCeDataReader^ sqlRead = command->ExecuteReader();
        group->Clear();
        while (sqlRead->Read()) {
            group->Add(gcnew Caller(
                System::Convert::ToInt32(sqlRead["id"]),
                System::Convert::ToString(sqlRead["FirstName"]),
                System::Convert::ToString(sqlRead["LastName"]),
                System::Convert::ToString(sqlRead["phone"]),
                System::Convert::ToInt32(sqlRead["categor"])
            ));
        }
        connect->Close();
        this->updateTable();
    }
```

```

    }
    else {
        connect->Open();
        SqlCommand^ command = connect->CreateCommand();
        command->CommandText = "SELECT * FROM Callers WHERE [category]=@group";
        command->Parameters->AddWithValue(
            "@group",
            this->groupId
        );
        SqlDataReader^ sqlRead = command->ExecuteReader();
        group->Clear();
        while (sqlRead->Read()) {
            group->Add(gcnew Caller(
                System::Convert::ToInt32(sqlRead["id"]),
                System::Convert::ToString(sqlRead["FirstName"]),
                System::Convert::ToString(sqlRead["LastName"]),
                System::Convert::ToString(sqlRead["phone"]),
                System::Convert::ToInt32(sqlRead["category"])
            ));
        }
        connect->Close();
        this->updateTable();
    }
}

int getSelectedGroupIndex()
{
    connect->Open();
    String^ query = "SELECT id FROM Categories WHERE category LIKE '" +
comboBox1->Text + "'";
    SqlCommand^ command = connect->CreateCommand();
    command->CommandText = query;
    SqlDataReader^ sqlRead = command->ExecuteReader();

    int result = -1;
    bool hasRow = sqlRead->Read();
    if (hasRow)
    {
        result = System::Convert::ToInt32(sqlRead["id"]);
    }
    connect->Close();
    sqlRead->Close();
    return result;
}

void getQueryCallers(String^ query) {
    connect->Open();
    SqlCommand^ command = connect->CreateCommand();
    command->CommandText = query;
    SqlDataReader^ sqlRead = command->ExecuteReader();
    group->Clear();
    while (sqlRead->Read()) {
        group->Add(gcnew Caller(
            System::Convert::ToInt32(sqlRead["id"]),
            System::Convert::ToString(sqlRead["FirstName"]),
            System::Convert::ToString(sqlRead["LastName"]),
            System::Convert::ToString(sqlRead["phone"]),
            System::Convert::ToInt32(sqlRead["category"])
        ));
    }
    sqlRead->Close();
    connect->Close();
    this->updateTable();
}

```



```

    }

    String^ getNameCategor(int id)
    {
        connect->Open();
        String^ query = "SELECT category FROM Categories WHERE id LIKE '" + id +
        "'";

        SqlCeCommand^ command = connect->CreateCommand();
        command->CommandText = query;
        SqlCeDataReader^ sqlRead = command->ExecuteReader();
        String^ result;
        bool hasRow = sqlRead->Read();
        if (hasRow) {
            result = System::Convert::ToString(sqlRead["category"]);
        }
        connect->Close();
        sqlRead->Close();
        return result;
    }

    void updateTable() {
        int i = 0;
        dataGridView1->Rows->Clear();
        for each (Caller ^ caller in group) {
            this->dataGridView1->Rows->Add();
            this->dataGridView1->Rows[i]->Cells[0]->Value = caller->getId();
            this->dataGridView1->Rows[i]->Cells[1]->Value = caller->getName();
            this->dataGridView1->Rows[i]->Cells[2]->Value = caller->getSurname();
            this->dataGridView1->Rows[i]->Cells[3]->Value = caller->
>getTelNumber();
            this->dataGridView1->Rows[i]->Cells[4]->Value = this->
>getNameCategor(caller->getCategory());
            ++i;
        }
    }
}

```

57. Запрограмувати кнопки **Додати запис**, **Редагувати запис**, **Видалити запис**, **Очистити фільтр** та процедуру обробки завантаження форми **Kontakts** відповідно до коду, наведеного нижче:

```
#pragma endregion
```

```
// Запрограмування кнопки Додати запис
```

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
    EditKontakt^ edit = gcnew EditKontakt(0, getSelectedGroupIndex());
    edit->ShowDialog();
    this->getCallers();
}

```

```
// Запрограмування кнопки Редагувати запис
```

```
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
    try
    {
        EditKontakt^ edit = gcnew
        EditKontakt(System::Convert::ToInt32(dataGridView1->Rows[dataGridView1->CurrentCell->
        >RowIndex]->Cells[0]->Value));
        edit->ShowDialog();
        this->getCallers();
    }
    catch (...) {
    }
}

```

```

        MessageBox::Show("Select correct item");
    }
}

// Запрограмування кнопки Видалити запис
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
    try {
        connect->Open();
        String^ query = "DELETE FROM Callers WHERE [id]=\"" +
System::Convert::ToString(dataGridView1->Rows[dataGridView1->CurrentCell->RowIndex]-
>Cells[0]->Value);
        SqlCeCommand^ command = gcnew SqlCeCommand(query, connect);
        if (command->ExecuteNonQuery() > 0) {
            MessageBox::Show("Deleted!");
        }
        connect->Close();
        this->getCallers();
    }
    catch (...) {
        MessageBox::Show("Choose correct item");
    }
}

// Запрограмування кнопки Очистити фільтр
private: System::Void button4_Click(System::Object^ sender, System::EventArgs^ e) {
    comboBox1->SelectedIndex = -1;
    textBox1->Text = "";
    textBox2->Text = "";
    this->getCallers();
}

// Запрограмування процедури завантаження форми
private: System::Void Kontakts_Load(System::Object^ sender, System::EventArgs^ e) {
    connect->Open();
    SqlCeCommand^ command = gcnew SqlCeCommand("SELECT category FROM Categories",
connect);
    SqlCeDataReader^ sqlRead = command->ExecuteReader();
    while (sqlRead->Read()) {
        comboBox1->Items->Add(sqlRead["category"]);
    }
    connect->Close();
    sqlRead->Close();
    if (comboBox1->Items->Count > 0)
    {
        comboBox1->SelectedIndex = -1;
        this->getCallers();
    }
}
}

```

58. Запрограмувати команди **Сортувати за прізвищем**, **Сортувати за номером телефону**, **Сортувати за порядковим номером**, **Всі контакти головного меню Дані форми Телефонний довідник**, по чергово двічі клікаючи на них та вписуючи у відповідні заготовки процедур їхньої обробки програмну реалізацію:

```

private: System::Void сортуватиЗаПрізвищемToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^ e) {
    String^ query = "SELECT * FROM Callers order by LastName;";
    this->getQueryCallers(query);
}

```

```
}
private: System::Void
сортуватиЗаНомеромТелефонуToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
    String^ query = "SELECT * FROM Callers order by phone;";
    this->getQueryCallers(query);
}
private: System::Void
сортуватиЗаПорядковимНомеромToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
    String^ query = "SELECT * FROM callers order by id;";
    this->getQueryCallers(query);
}
private: System::Void всіКонтактиToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
    comboBox1->SelectedIndex = -1;
    this->getCallers();
}
```

59. Запрограмувати процедуру вибору номера категорії за допомогою компоненти **ComboBox**, двічі клікнувши по ній і вписавши відповідний код:

```
private: System::Void comboBox1_SelectedIndexChanged(System::Object^ sender,
System::EventArgs^ e) {
    this->getCallers();
}
```

60. Визначити метод пошуку даних за вказаним критерієм:

```
private: System::Void search(System::Object^ sender, System::EventArgs^ e) {
    String^ query = "SELECT * FROM Callers WHERE [LastName] LIKE '%" + textBox1->Text + "%' and [phone] LIKE '%" + textBox2->Text + "%'";
    this->getQueryCallers(query);
}
```

61. У вікні властивостей для об'єктів **TextBox1** та **TextBox2** на закладці **События** у властивості **TextChanged** змінити значення на виклик методу **search()**.

62. Побудувати рішення **Построение – Построить решение** та запустити програму на виконання.

Лабораторне завдання

1. Створити на комп'ютері програмний проєкт у середовищі Visual C++ для відображення та обробки даних БД(база даних з попередньої роботи).
2. У протоколі лабораторної роботи надати скріншоти інтерфейсу користувача, програмні коди з описом їх використання та відповіді на контрольні питання.
3. Здавати звіт у форматі *pdf і з посиланням на архів проєкту.

Контрольні запитання

1. На використанні яких компонентів заснований доступ до даних в ADO.NET
2. Що таке провайдер
3. З чого складається провайдер даних
4. Які об'єкти використовують для обробки бази даних