ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

C#. WPF. Робота з SQLite

Лекція 15

План

1. Що таке SQLite?
2. Підключення SQLite
3. Операції CRUD
4. Приклад

Що таке SQLite?

SQLite - це компактна, вбудована система керування реляційними базами даних (СУБД). Вона відрізняється від багатьох інших СУБД, таких як MySQL або PostgreSQL, тим, що її вбудовують безпосередньо в додаток, не вимагаючи окремого сервера баз даних.

Основна характеристика SQLite

1. SQLite дуже компактна і легка у використанні. Не потребує великих ресурсів системи і може працювати практично на всіх платформах, включно з мобільними пристроями та вбудованими системами.

2. SQLite інтегрується безпосередньо в додаток. Файл бази даних SQLite зберігається на диску, і додаток працює з ним безпосередньо.

3. SQLite поширюється як бібліотека з відкритим вихідним кодом (open source). Можна змінювати вихідний код відповідно потреб.

Основна характеристика SQLite

4. SQLite підтримує повний набір SQL-команд. Можна виконувати операції вставки, вибірки, оновлення та видалення даних з використанням SQL-запитів.

5. Оскільки SQLite вбудовується у додаток і не потребує окремого сервера, адміністрування бази даних зводиться до мінімуму.

6.SQLite підтримує транзакції, що гарантує цілісність даних і забезпечує атомарність операцій.

7. SQLite-базу можна перенести з однієї платформи на іншу без зміни коду програми.

Встановлення необхідного NuGet-пакета

- У власному проекті перейдіть у Tools -> NuGet Package Manager -> Manage NuGet Packages for Solution.

- У вкладці Browse знайдіть та встановіть пакет System.Data.SQLite. Це надасть необхідні інструменти для роботи з SQLite.

- Імпортування простору імен. У файлі C#, де ви плануєте працювати з базою даних, додайте:

```
using System.Data.SQLite;
```

Підключення SQLite

- Для з'єднання з базою даних SQLite використовується об'єкт SQLiteConnection. Рядок з'єднання містить інформацію про місцезнаходження файлу бази даних.

```
string connectionString = "Data Source=mydatabase.db; Version=3;";
SQLiteConnection connection = new SQLiteConnection(connectionString);
```

Data Source - ім'я файлу бази даних. Якщо файл не існує, він буде створений.
Version - зазвичай встановлюється на 3 для SQLite.

- Перед виконанням будь-яких операцій з базою даних, з'єднання потрібно відкрити:

```
connection.Open();
```

Створення таблиці

Для створення таблиці використовується об'єкт SQLiteCommand та SQL-запит CREATE TABLE.

```
SQLiteCommand command = new SQLiteCommand("CREATE TABLE IF NOT EXISTS Users (Id INTEGER PRIMARY KEY, Name TEXT, .....)", connection);
command.ExecuteNonQuery();
```

- CREATE TABLE IF NOT EXISTS - створює таблицю, якщо вона не існує.
- Users - назва таблиці.
- Id INTEGER PRIMARY KEY, Name TEXT, ... - стовпці таблиці.

Створення таблиці

Нехай є таблиця Cars зі стовпцями:

- Id (INTEGER PRIMARY KEY AUTOINCREMENT),
- Make (TEXT),
- Model (TEXT),
- Year (INTEGER),
- Color (TEXT).

```
string createTableQuery = @"
    CREATE TABLE IF NOT EXISTS Cars (
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Make TEXT NOT NULL,
        Model TEXT NOT NULL,
        Year INTEGER NOT NULL,
        Color TEXT
    )";
```

Операції з даними (CRUD)

Операції CRUD (Create, Read, Update, Delete) реалізуються за допомогою SQL-запитів, які виконуються через бібліотеку System.Data.SQLite.

- Create - створення, додавання нового запису:

INSERT INTO Cars (Перелік стовпців) VALUES (Значення, які будуть вставлені у відповідні стовпці. Використання @ перед назвою параметра є ознакою параметризованого запиту.)

```
string insertQuery = @"
        INSERT INTO Cars (Make, Model, Year, Color)
        VALUES (@Make, @Model, @Year, @Color)";
```

Операції з даними (CRUD)

```
string insertQuery = @"
        INSERT INTO Cars (Make, Model, Year, Color)
        VALUES (@Make, @Model, @Year, @Color)";

using (SQLiteConnection connection = new SQLiteConnection($"Data Source = {DatabaseName}; Version=3;"))
{
    connection.Open();
    using (SQLiteCommand command = new SQLiteCommand(insertQuery, connection))
    {
        // Використання параметрів для запобігання SQL-ін'єкціям
        command.Parameters.AddWithValue("@Make", make);
        command.Parameters.AddWithValue("@Model", model);
        command.Parameters.AddWithValue("@Year", year);
        command.Parameters.AddWithValue("@Color", color);

        command.ExecuteNonQuery(); // Виконує запит на вставку
    }
}
```

Операції з даними (CRUD)

- **Read** (читання, отримання) даних:

```
  string selectQuery = "SELECT Id, Make, Model, Year, Color FROM Cars";
using (SQLiteCommand command = new SQLiteCommand(selectQuery, connection)){
   using (SQLiteDataReader reader = command.ExecuteReader()) {
      while (reader.Read())
      {
         _cars.Add(new Car
         {
            Id = reader.GetInt32(0),
            Make = reader.GetString(1),
            Model = reader.GetString(2),
            Year = reader.GetInt32(3),
            Color = reader.IsDBNull(4) ? null : reader.GetString(4)
         });
      }
   }
}
```

методи для отримання значення відповідного стовпця поточного рядка за його індексом (починаючи з 0)

перевіряє, чи є значення стовпця NULL у базі даних

Операції з даними (CRUD)

- Update (оновлення, зміна існуючого запису):

```
string updateQuery = @"
            UPDATE Cars
    SET Make = @Make, Model = @Model, Year = @Year, Color = @Color
    WHERE Id = @Id";

using (SQLiteCommand command = new SQLiteCommand(updateQuery, connection)) {
    command.Parameters.AddWithValue("@Id", updatedCar.Id);
    command.Parameters.AddWithValue("@Make", updatedCar.Make);
    ...
    command.ExecuteNonQuery();
}
```

UPDATE Cars SET ... WHERE ... - синтаксис для оновлення даних у таблиці Cars.
SET Make = @Make, Model = @Model, ... - перелік стовпців, які потрібно оновити, та їх нові значення.
WHERE Id = @Id - умова, яка визначає, який саме запис потрібно оновити (зазвичай за унікальним ідентифікатором).

Операції з даними (CRUD)

- Delete - видалення існуючого запису:

```
string deleteQuery = "DELETE FROM Cars WHERE Id = @Id";
using (SQLiteCommand command = new SQLiteCommand(deleteQuery, connection))
{
    command.Parameters.AddWithValue("@Id", selectedCar.Id);
    command.ExecuteNonQuery();
}
```

DELETE FROM Cars WHERE ... - основний SQL-синтаксис для видалення даних з таблиці Cars.
WHERE Id = @Id - умова, яка визначає, який саме запис потрібно видалити.

Операції з даними (CRUD)

- Додавання даних

```
command = new SQLiteCommand("INSERT INTO Users VALUES (' Id ', ' Name ')", connection);
```

- Вибір даних

```
command = new SQLiteCommand("SELECT * FROM Users", connection);
SQLiteDataReader reader = command.ExecuteReader();
        while (reader.Read()){
            Console.WriteLine($"Id: {reader["Id"]}, Name: {reader["Name"]}, ... }");
        }
```

Операції з даними (CRUD)

- Оновлення даних

```
command = new SQLiteCommand("UPDATE Users SET Name = 'Name' WHERE Id = 1", connection);
```

- Видалення даних

```
command = new SQLiteCommand("DELETE FROM Users WHERE Id = 1", connection);
```

- Закриття з'єднання після завершення роботи з базою даних з'єднання потрібно закрити:

```
connection.Close();
```

Приклад

```
public class Car : INotifyPropertyChanged
{
    private int _id;
    private string _make;
    private string _model;
    private int _year;
    private string _color;

    public int Id{
       get { return _id; }
       set { _id = value; OnPropertyChanged(nameof(Id));}
    }

    public string Make{
       get { return _make; }
       set {_make = value; OnPropertyChanged(nameof(Make));
       }
    }

...
```

Клас реалізує INotifyPropertyChanged для підтримки одностороннього та двостороннього зв'язування (елемент інтерфейсу оновлюється, коли джерело динамічно змінено)

Викликається OnPropertyChanged при кожному оновленні властивості

Приклад

```
public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Метод OnPropertyChanged(string propertyName) - захищений віртуальний метод викликає подію PropertyChanged для вказаної властивості.

Приклад

INotifyPropertyChanged - інтерфейс реалізується для забезпечення можливості двостороннього прив'язування даних (data binding) у WPF. Коли значення властивості змінюється, викликається подія PropertyChanged, яка повідомляє інтерфейс користувача про необхідність оновлення відображення.

Приклад

```
public partial class MainWindow : Window
{
    private const string DatabaseName = "CarsDatabase.sqlite";
    private readonly List<Car> _cars = new List<Car>();
    public MainWindow()
    {
        InitializeComponent();
        DataContext = this;
        InitializeDatabase();
        LoadCars();
    }
    public List<Car> Cars => _cars;
```

Клас MainWindow – Головне вікно

DatabaseName визначає ім'я файлу бази даних SQLite .

_cars - список об'єктів Car, який використовується для зберігання завантажених з бази даних автомобілів та є джерелом даних для елемента ListView у WPF.

Конструктор MainWindow()

InitializeComponent() - ініціалізація компонентів інтерфейсу користувача, описаних у файлі MainWindow.xaml.

DataContext = this - встановлює поточний екземпляр MainWindow як контекст даних для вікна. Це дозволяє елементам керування у XAML прив'язуватися до властивостей та команд класу MainWindow (наприклад, до списку Cars).

InitializeDatabase() - викликає метод для створення бази даних та таблиці Cars, якщо її ще не існують.

LoadCars() -викликає метод для завантаження даних про автомобілі з бази даних до списку _cars.

Властивість Cars повертає приватний список _cars, використовується для прив'язування даних у XAML

Приклад

```
private void InitializeDatabase() {
  if (!File.Exists(DatabaseName)) {
    using (SQLiteConnection connection = new SQLiteConnection($"Data Source={DatabaseName};Version=3;")) {
      connection.Open();

      string createTableQuery = @"
        CREATE TABLE IF NOT EXISTS Cars (
          Id INTEGER PRIMARY KEY AUTOINCREMENT,
          Make TEXT NOT NULL,
          Model TEXT NOT NULL,
          Year INTEGER NOT NULL,
          Color TEXT
        )";

      using (SQLiteCommand command = new SQLiteCommand(createTableQuery, connection))
      {
        command.ExecuteNonQuery();
      }
    }
  }
}
```

Перевіряє, чи існує файл бази даних (CarsDatabase.sqlite)

Створює нове з'єднання з базою даних SQLite.

Приклад

```
private void LoadCars() {
    _cars.Clear();
    using (SQLiteConnection connection = new SQLiteConnection($"Data Source={DatabaseName};Version=3;")) {
        connection.Open();

        string selectQuery = "SELECT Id, Make, Model, Year, Color FROM Cars";
        using (SQLiteCommand command = new SQLiteCommand(selectQuery, connection))  {
            using (SQLiteDataReader reader = command.ExecuteReader())  {
                while (reader.Read()) {
                    _cars.Add(new Car {
                        Id = reader.GetInt32(0),
                        Make = reader.GetString(1),
                        Model = reader.GetString(2),
                        Year = reader.GetInt32(3),
                        Color = reader.IsDBNull(4) ? null : reader.GetString(4)
                    });
                }
            }
        }
    }
    CarsListView.Items.Refresh();
}
```

Приклад

```
private void AddCarButton_Click(object sender, RoutedEventArgs e){
    var addCarWindow = new AddCarWindow();
    if (addCarWindow.ShowDialog() == true) {
        var newCar = addCarWindow.NewCar;
        using (SQLiteConnection connection = new SQLiteConnection($"Data Source={DatabaseName};Version=3;"))  {
            connection.Open();

            string insertQuery = @"
                INSERT INTO Cars (Make, Model, Year, Color)
                VALUES (@Make, @Model, @Year, @Color)";

            using (SQLiteCommand command = new SQLiteCommand(insertQuery, connection)) {
                command.Parameters.AddWithValue("@Make", newCar.Make);
                command.Parameters.AddWithValue("@Model", newCar.Model);
                command.Parameters.AddWithValue("@Year", newCar.Year);
                command.Parameters.AddWithValue("@Color", newCar.Color);
                command.ExecuteNonQuery();
            }
        }
        LoadCars();
    }
}
```

Приклад

Загальний принцип роботи програми:

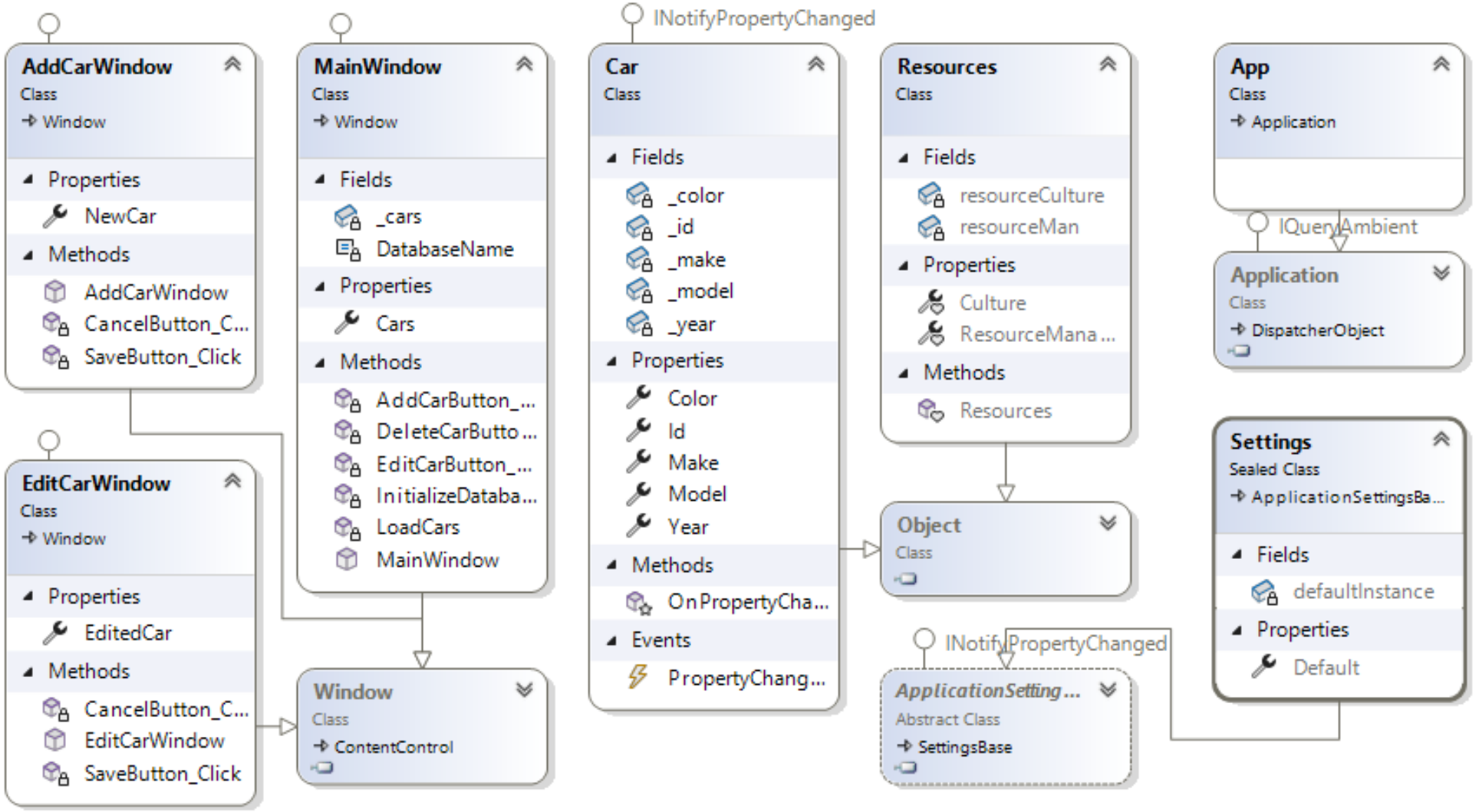
При запуску головного вікна (MainWindow) ініціалізується база даних SQLite (створюється файл та таблиця, якщо їх немає). Завантажуються дані про автомобілі з бази даних та відображаються у ListView.

При натисканні кнопки «Додати» відкривається вікно AddCarWindow. Користувач вводить дані про новий автомобіль. При натисканні «Зберегти» дані з прив'язаного об'єкта NewCar записуються до бази даних, і список автомобілів у головному вікні оновлюється.

При виборі автомобіля у ListView та натисканні кнопки «Видалити», обраний автомобіль видаляється з бази даних, і список оновлюється.

При виборі автомобіля та натисканні кнопки «Редагувати» відкривається вікно EditCarWindow з попередньо заповненими даними обраного автомобіля. Користувач може редагувати дані. При натисканні «Зберегти» оновлені дані записуються до бази даних, і список оновлюється.

Приклад


AddCarWindow
Class
Window
Properties
NewCar
Methods
AddCarWindow
CancelButton_C...
SaveButton_Click
MainWindow
Class
Window
Fields
_cars
DatabaseName
Properties
Cars
Methods
AddCarButton_...
DeleteCarButto...
EditCarButton_...
InitializeDataba...
LoadCars
MainWindow
INotifyPropertyChanged
Car
Class
Fields
_color
_id
_make
_model
_year
Properties
Color
Id
Make
Model
Year
Methods
OnPropertyCha...
Events
PropertyChang...
Resources
Class
Fields
resourceCulture
resourceMan
Properties
Culture
ResourceMana...
Methods
Resources
App
Class
Application
IQueryAmbient
Application
Class
DispatcherObject
EditCarWindow
Class
Window
Properties
EditedCar
Methods
CancelButton_C...
EditCarWindow
SaveButton_Click
Window
Class
ContentControl
Object
Class
Settings
Sealed Class
ApplicationSettingsBa...
Fields
defaultInstance
Properties
Default
INotifyPropertyChanged
ApplicationSetting...
Abstract Class
SettingsBase

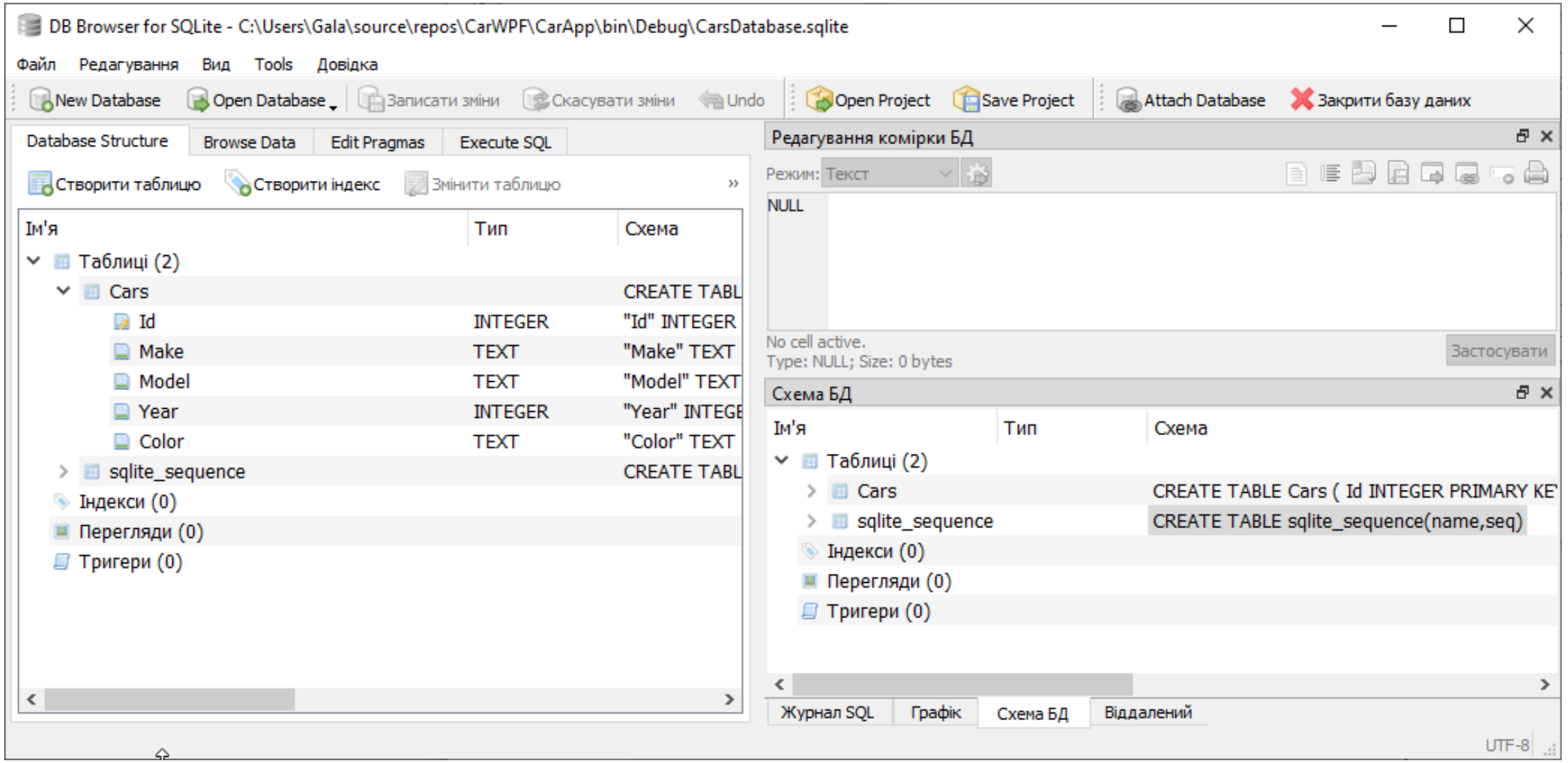
Браузер баз даних для SQLite

DB Browser for SQLite (DB4S) – це високоякісний візуальний інструмент з відкритим кодом , розроблений для тих, хто хоче створювати, шукати та редагувати файли баз даних SQLite або SQLCipher .

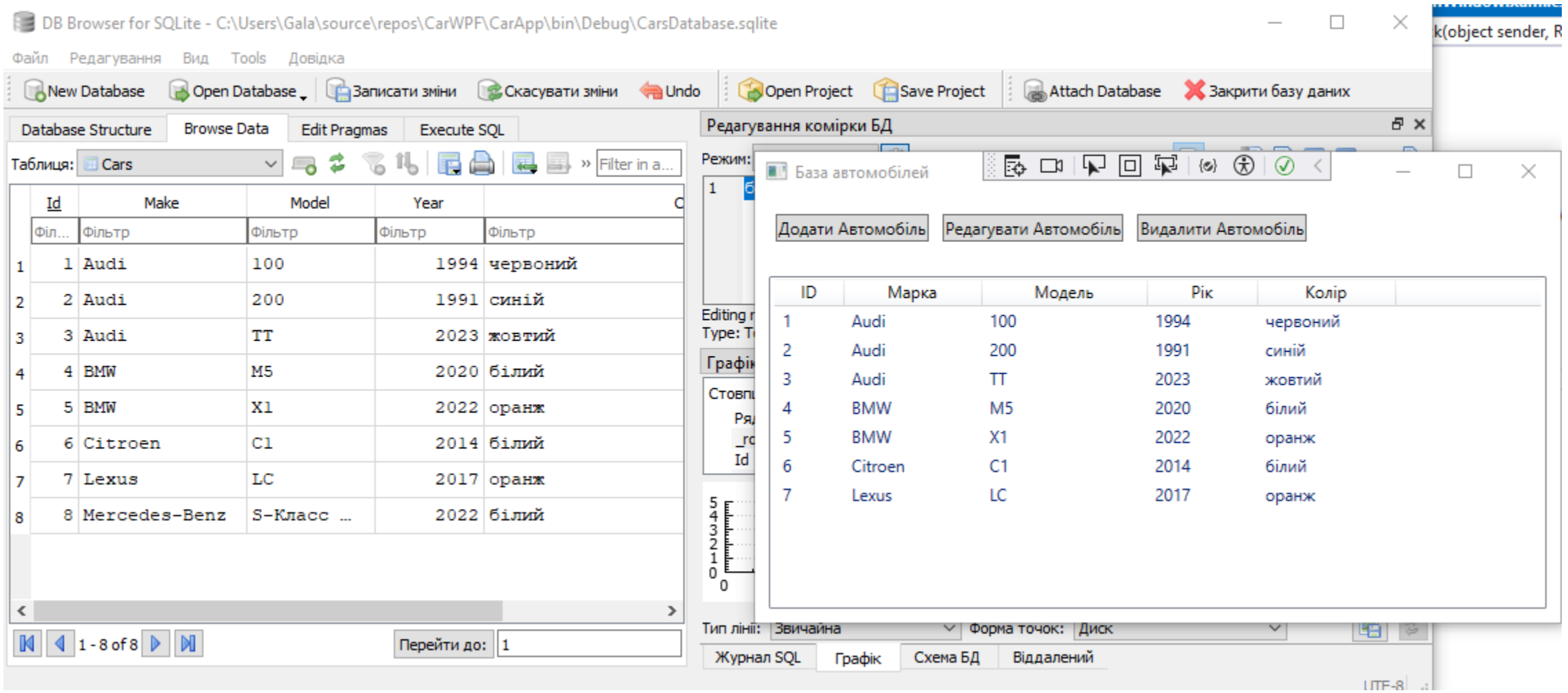
DB4S надає звичний інтерфейс бази даних, схожий на електронні таблиці, а також повноцінний засіб SQL-запитів.

https://sqlitebrowser.org/

Браузер баз даних для SQLite



Браузер баз даних для SQLite



Важливі моменти

- ✓ Використання блоків using гарантує автоматичне закриття та звільнення ресурсів після їх використання, навіть якщо виникне виняткова ситуація.
- ✓ Використовуйте параметризовані запити (@ParameterName та command.Parameters.AddWithValue()) для передачі значень у SQL-запити. Це є важливим заходом безпеки для запобігання SQL-ін'єкціям.
- ✓ У реальних застосунках необхідно додавати блоки try-catch для обробки можливих помилок під час роботи з базою даних.
- ✓ Для покращення продуктивності WPF-застосунків, особливо при тривалих операціях з базою даних, розгляньте можливість використання асинхронних методів (ExecuteNonQueryAsync, ExecuteReaderAsync).