Київський національний університет імені Тараса Шевченка

ЗВІТ

до Етапу 2. Розробка десктоп-версії (десктоп-клієнта)
практичного завдання (модуль 1)
з дисципліни “Інформаційні технології”
на тему:

Часткова реалізація системи управління табличними базами даних

Виконав
Студент четвертого курсу
Групи ТТП-41
Факультету комп’ютерних наук та кібернетики
Назарій ЯГОТІН

Київ, 2025

Зміст

1 Постановка задачі

1.1 Мета роботи

Розробити десктоп-застосунок для управління табличними базами даних з підтримкою базових типів даних та двох спеціалізованих типів відповідно до варіанту.

1.2 Основні вимоги

Система повинна забезпечувати:

- Створення бази даних з необмеженою кількістю таблиць
- Підтримку базових типів даних: `integer`, `real`, `char`, `string`
- Підтримку додаткових типів: **інтервал цілих чисел** та **текстові файли**
- Створення та знищення таблиць з валідацією
- CRUD операції над рядками таблиць
- Збереження та завантаження бази даних з диску
- Додаткову операцію: **перейменування та перестановку колонок**

1.3 Додаткові типи даних

Інтервал цілих чисел (integerInvl): представляє діапазон значень з мінімумом та максимумом. Використовується для зберігання числових інтервалів, наприклад, вікових груп, діапазонів температур тощо.

Текстові файли: дозволяє зберігати посилання на текстові файли з їх метаданими (ім'я файлу, розмір, MIME-тип). Фактичний вміст файлів зберігається окремо у файловій системі.

2 Використані технології

2.1 Мови програмування та фреймворки

- **C# 14.0** — основна мова розробки
- **.NET 9.0** — платформа для виконання
- **WPF (Windows Presentation Foundation)** — фреймворк для графічного інтерфейсу
- **NUnit** — фреймворк для юніт - тестів
- **XAML** — мова розмітки для UI

2.2 Архітектурні рішення

Проєкт організовано за багатошаровою архітектурою:

- **CoreLib** — бібліотека класів з бізнес-логікою
  - `Models` — моделі даних (Database, Table, Column)
  - `Services` — сервіси для роботи з даними
  - `Serialization` — конвертери для JSON
  - `Common` — допоміжні класи та утиліти
- **DesktopApp** — WPF застосунок (presentation layer)
- **Tests** — модульні тести (NUnit)

2.3 Бібліотеки та інструменти

- **System.Text.Json** — серіалізація/десеріалізація БД
- **NUnit** — фреймворк для unit-тестування
- **Microsoft.Win32** — діалоги відкриття/збереження файлів

3 Початок роботи з застосунком

3.1 Головне вікно застосунку

При запуску застосунку відкривається головне вікно з меню та робочою областю.

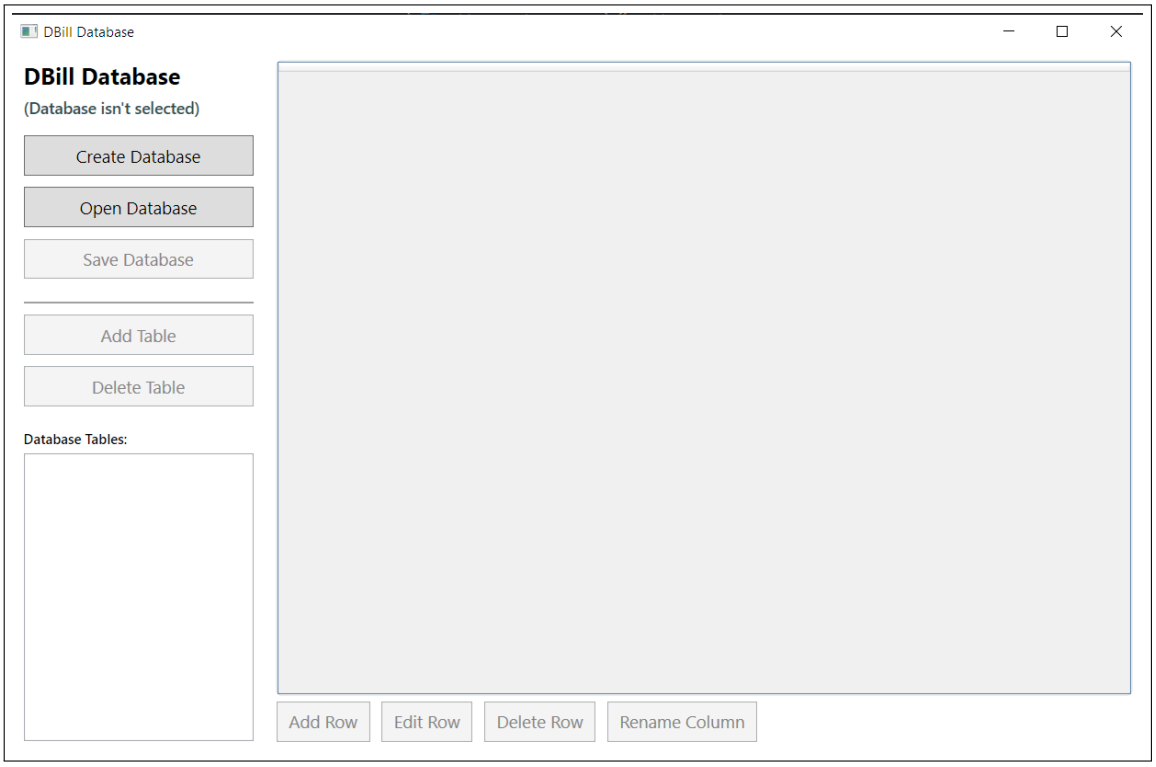


Рис. 1: Головне вікно застосунку

Головне вікно містить:

- **Меню** — доступ до основних операцій (Create/Open/Save Database, Add/Delete Table)
- **Список таблиць** — відображає всі таблиці поточної БД
- **Область даних** — показує вміст обраної таблиці
- **Панель інструментів** — швидкий доступ до операцій

3.2 Створення нової бази даних

Для створення нової БД використовується кнопка `Create Database`. Користувач вводить ім'я бази даних у діалоговому вікні.

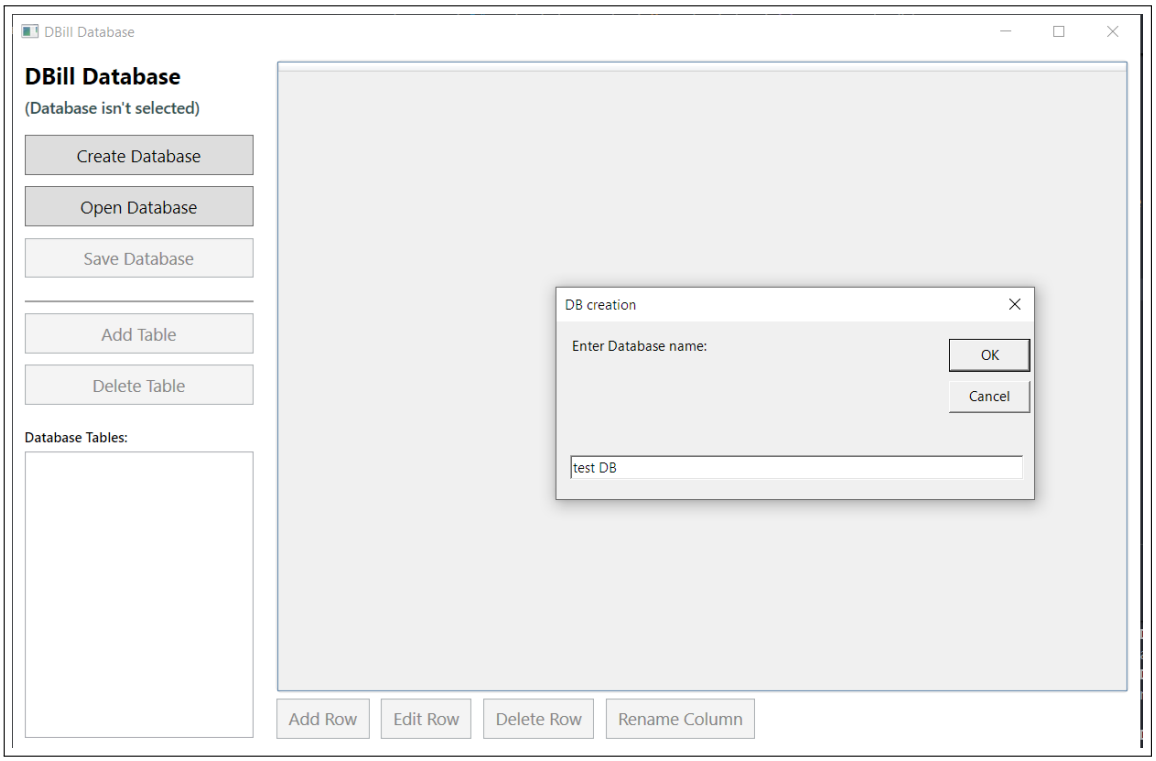


Рис. 2: Діалогове вікно створення бази даних

4 Створення БД і таблиць

4.1 Діалог створення таблиці

Для створення таблиці використовується діалогове вікно, яке дозволяє:

- Задати ім'я таблиці
- Додати колонки з вказанням імені та типу
- Видалити колонки

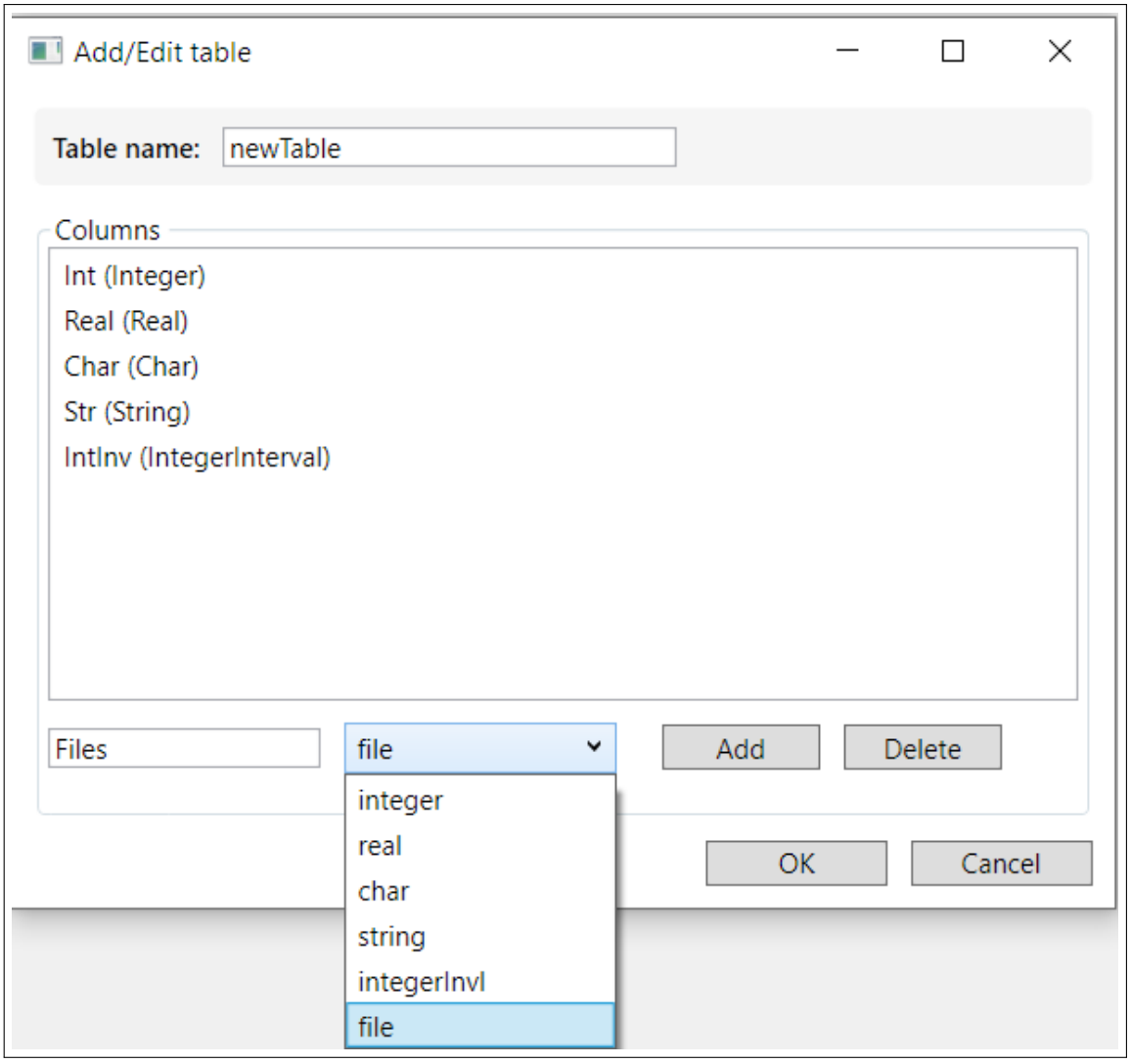


Рис. 3: Діалогове вікно створення таблиці

4.2 Підтримувані типи даних

При створенні колонки доступні наступні типи:

- `Integer` — цілі числа
- `Real` — дійсні числа
- `Char` — один символ
- `String` — рядок тексту
- `IntegerInterval` — інтервал цілих чисел
- `TextFile` — текстовий файл

4.3 Валідація схеми таблиці

Система автоматично перевіряє:

- Унікальність та коректність імені таблиці
- Коректність імен колонок
- Наявність хоча б однієї колонки
- Унікальність імен колонок у межах таблиці

5 Робота з таблицями і даними

5.1 Перегляд даних таблиці

При виборі таблиці зі списку, у правій частині відображаються всі рядки у вигляді таблиці.

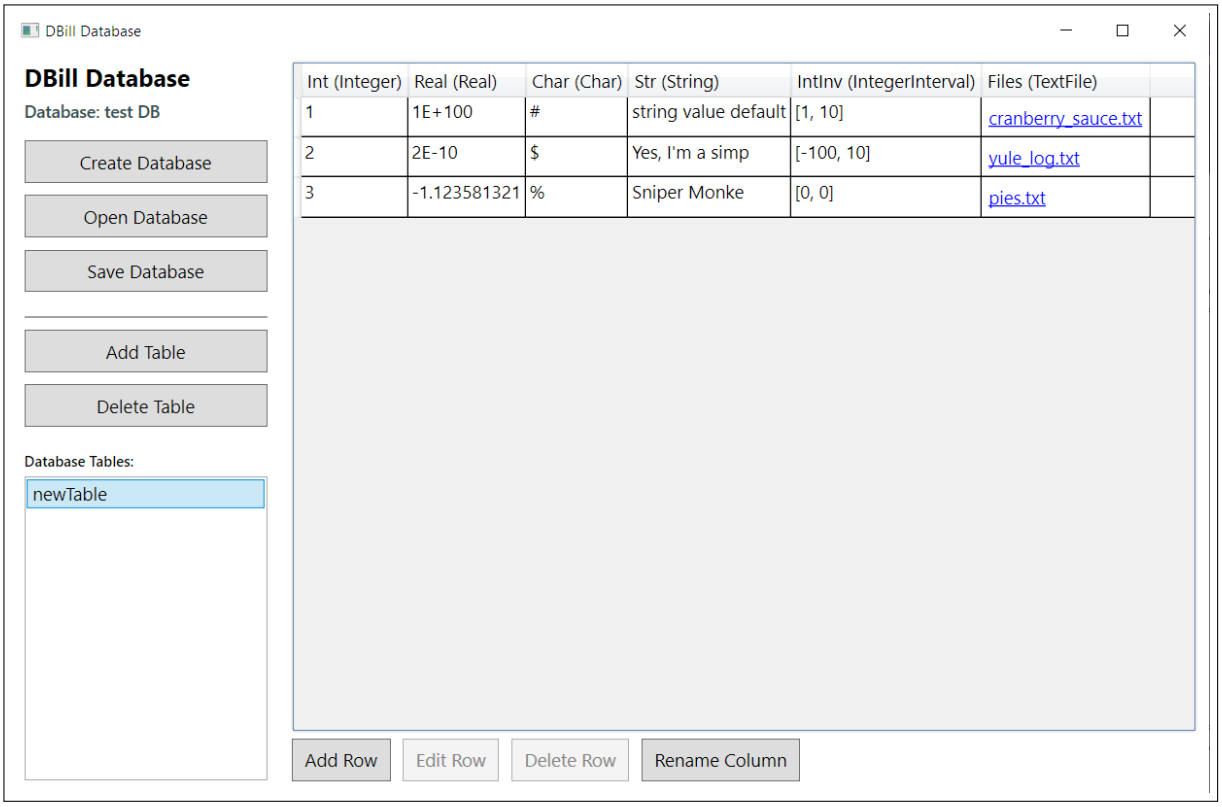


Рис. 4: Відображення даних таблиці

5.2 Додавання рядка

Для додавання нового рядка відкривається діалогове вікно з полями для кожної колонки. Інтерфейс адаптується залежно від типу даних:

- Для базових типів — текстове поле
- Для інтервалу — два поля (мін/макс)
- Для файлів — кнопка вибору файлу

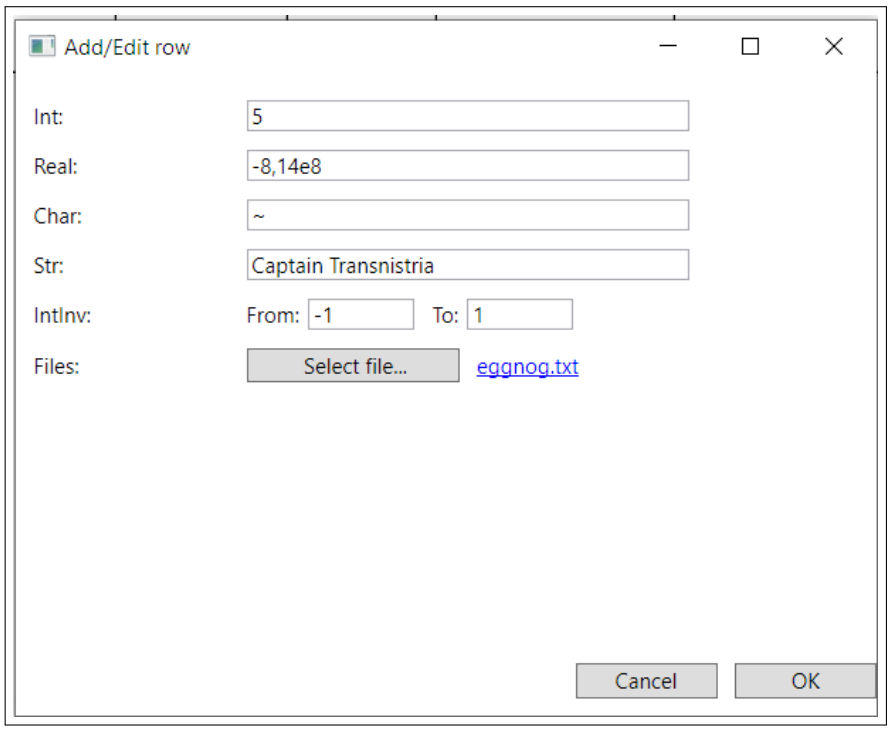


Рис. 5: Діалогове вікно додавання рядка

5.3 Редагування рядка

Подвійний клік по рядку відкриває той самий діалог, але з попередньо заповненими даними.

5.4 Видалення рядка

Видалення рядка виконується через кнопку на панелі інструментів.

5.5 Валідація даних при введенні

При збереженні рядка система перевіряє коректність введених даних. Реалізація валідації виконується у сервісі TableService:

```
public (Dictionary<string, object?>, ValidationResult)
    ParseAndValidateRowData(string tableName, Dictionary<string, object?>
    ↪ rawData, Dictionary<string, FileRecord?> fileRecords)
{
    var table = _databaseService.GetTable(tableName);
    var parsedValues = new Dictionary<string, object?>();
    var errors = new List<string>();

    foreach (var column in table.Columns)
    {
        switch (column.Type)
        {
            case DataType.Integer:
                break;
            case DataType.IntegerInterval:
                break;
            // Інші типи...
        }
    }

    return (parsedValues, new ValidationResult(errors));
}
```

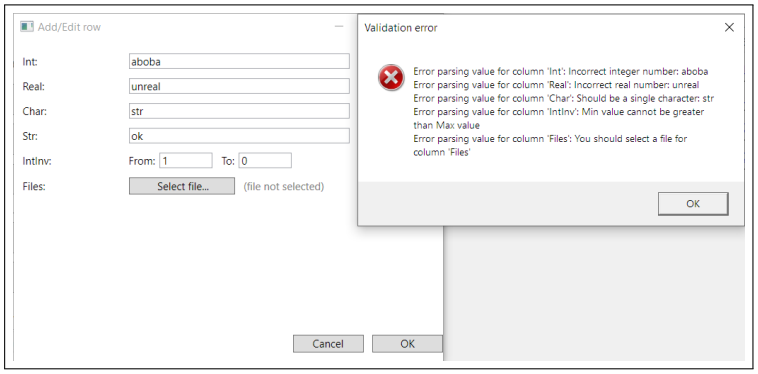
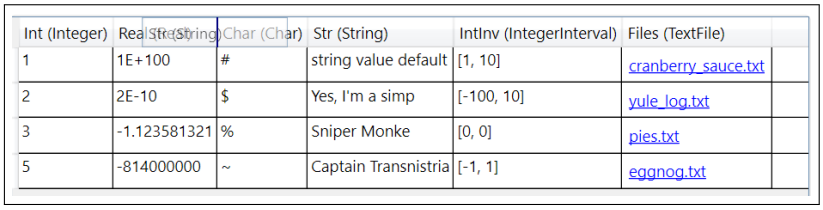


Рис. 6: Повідомлення про помилку валідації

5.6 Перейменування та перестановка колонок

Додаткова операція згідно варіанту — можливість перейменувати колонку або змінити її позицію у таблиці.

- **Перестановка колонок** відбувається затисканням відповідного хедера у графічному представленні таблиці та пересуванням його курсором на бажане місце
- **Перейменування колонки** відбувається у відповідному діалозі після натискання на кнопку



Int (Integer)	Real (Real)	Char (Char)	Str (String)	IntInv (IntegerInterval)	Files (TextFile)
1	1E+100	#	string value default	[1, 10]	cranberry_sauce.txt
2	2E-10	$	Yes, I'm a simp	[-100, 10]	yule_log.txt
3	-1.123581321	%	Sniper Monke	[0, 0]	pies.txt
5	-814000000	~	Captain Transnistria	[-1, 1]	eggnog.txt

Рис. 7: Процес перестановки колонок

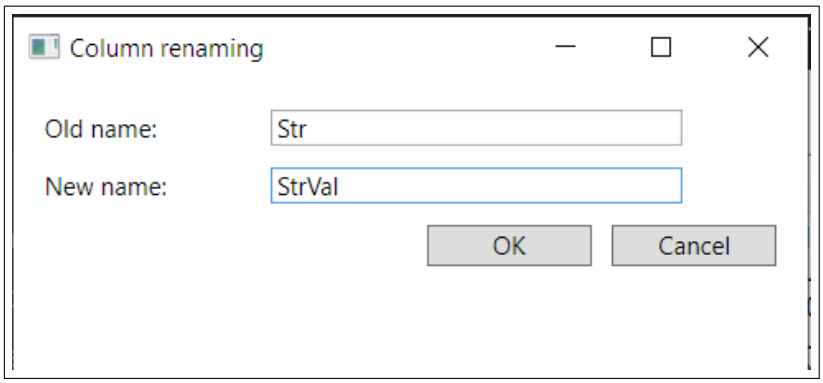


Рис. 8: Діалог перейменування колонки

Int (Integer)	Real (Real)	StrVal (String)	Char (Char)	IntInv (IntegerInterval)	Files (TextFile)
1	1E+100	string value default	#	[1, 10]	cranberry_sauce.txt
2	2E-10	Yes, I'm a simp	$	[-100, 10]	yule_log.txt
3	-1.123581321	Sniper Monke	%	[0, 0]	pies.txt
5	-814000000	Captain Transnistria	~	[-1, 1]	eggnog.txt

Рис. 9: Результат перейменування та перестановки колонок

6 Збереження БД на пристрій та завантаження БД

6.1 Збереження бази даних

БД зберігається у форматі JSON. Процес збереження включає:

1. Серіалізацію структури БД
2. Збереження метаданих файлів
3. Запис JSON-файлу на диск

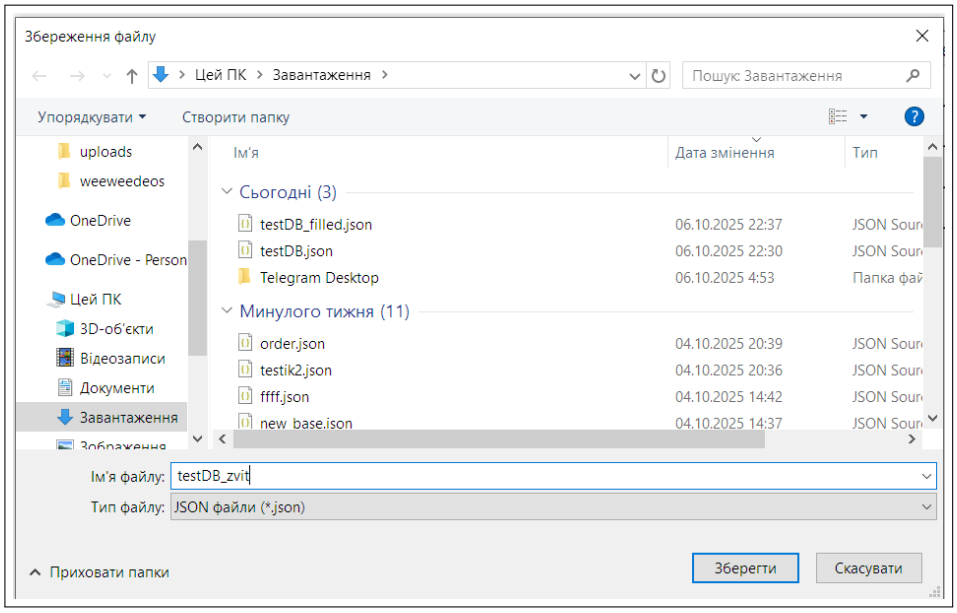


Рис. 10: Діалог збереження бази даних

Основний код збереження у `DatabaseService`:

```
public async Task SaveDatabaseAsync(string filePath)
{
    if (_currentDatabase == null)
        throw new InvalidOperationException("No database is currently loaded");

    var validation = _currentDatabase.Validate();
    if (!validation.IsValid)
        throw new InvalidOperationException($"Cannot save invalid database:
        ↪  {string.Join(", ", validation.Errors)}");

    await _storageService.SaveDatabaseAsync(_currentDatabase, filePath);
}
```

6.2 Завантаження бази даних

При завантаженні БД система використовує тимчасове сховище для безпечної валідації:

1. Завантаження у тимчасове сховище
2. Валідація структури БД
3. Перевірка цілісності файлів
4. Копіювання файлів у основне сховище
5. Заміна поточної БД

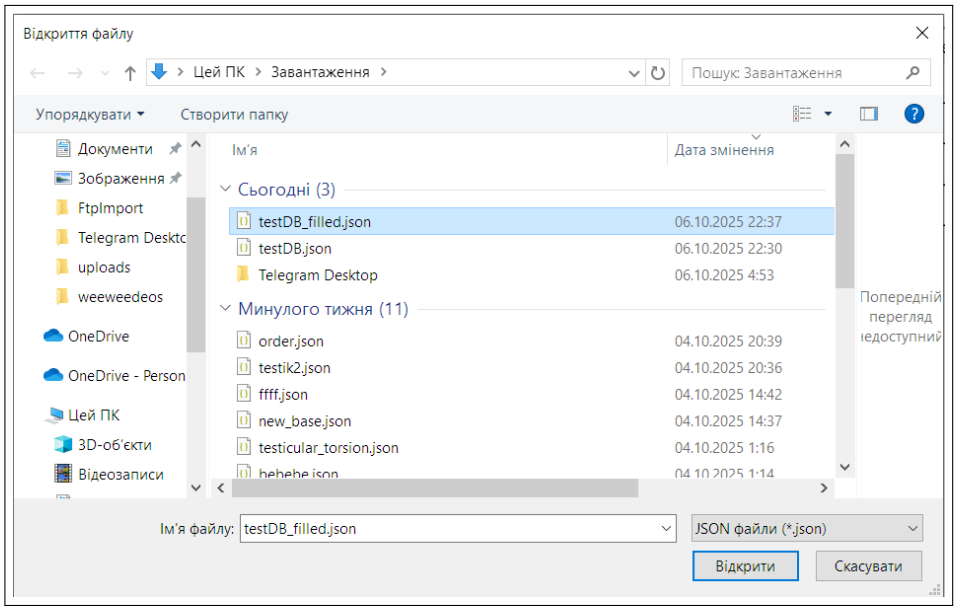


Рис. 11: Діалог відкриття бази даних

Ключова частина логіки безпечного завантаження:

```
public async Task<Database> LoadDatabaseAsync(string filePath)
{
    // Завантажуємо в тимчасове сховище
    var tempDatabase = await _tempStorageService
        .LoadDatabaseAsync(filePath);

    // Валідація
    var validation = tempDatabase.Validate();
    if (!validation.IsValid)
    {
        await _tempFileService.CleanupAllFilesAsync();
        throw new InvalidOperationException(
            $"Database validation failed");
    }

    // Зберігаємо файли на диск у тимчасову папку
    await SaveFileContentsForDatabaseAsync(
        tempDatabase, _tempFileService);

    // Якщо все успішно - очищаємо стару базу
    await CloseDatabase();

    // Копіюємо файли з тимчасової папки в основну
    await CopyFilesFromTempToMainAsync(tempDatabase);

    _currentDatabase = tempDatabase;
    return _currentDatabase;
}
```

6.3 Формат збереження

БД зберігається як JSON-файл з повними вмістом файлів у форматі base64. Файли таблиць завантаженої / поточної бази зберігаються окремо у папці `uploads/`. При роботі з БД в оперативній пам'яті зберігається лише шлях до файлу

7 Інтервальний тип цілих чисел

7.1 Модель даних

Інтервал представлено класом `IntegerInterval`:

```
public class IntegerInterval
{
    public int Min { get; set; }
    public int Max { get; set; }

    [JsonConstructor]
    public IntegerInterval(int min, int max)
    {
        if (min > max)
            throw new ArgumentException("Min value cannot be greater than Max
            ↪  value");

        Min = min;
        Max = max;
    }

    public bool Contains(int value)
    {
        return value >= Min && value <= Max;
    }
}
```

7.2 Інтерфейс для введення інтервалу

При додаванні/редагуванні рядка для колонки типу `IntegerInterval` відображаються два поля: **From** та **To**.

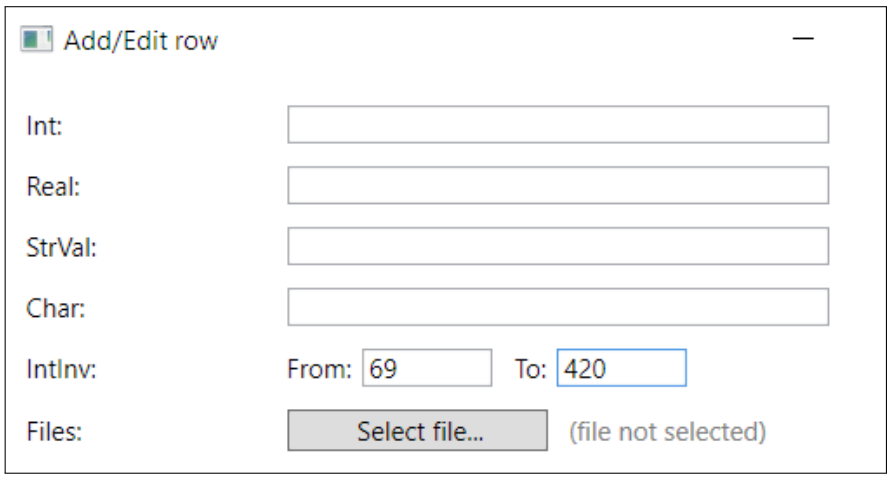


Рис. 12: Введення значення інтервального типу

7.3 Валідація інтервалу

Система перевіряє:

- Коректність числових значень
- Умову $Min \leq Max$
- Відповідність типу `integer` для обох меж

7.4 Серіалізація інтервалу

Для збереження у JSON використовується спеціальний конвертер:

```
public class IntegerIntervalJsonConverter :
    JsonConverter<IntegerInterval>
{
    public override IntegerInterval Read(
        ref Utf8JsonReader reader,
        Type typeToConvert,
        JsonSerializerOptions options)
    {
        var obj = JsonSerializer
            .Deserialize<Dictionary<string, int>>(
                ref reader, options);
        return new IntegerInterval(
            obj["Min"], obj["Max"]);
    }

    public override void Write(
        Utf8JsonWriter writer,
        IntegerInterval value,
        JsonSerializerOptions options)
    {
        writer.WriteStartObject();
        writer.WriteNumber("Min", value.Min);
        writer.WriteNumber("Max", value.Max);
        writer.WriteEndObject();
    }
}
```

8 Робота з текстовими файлами

8.1 Модель FileRecord

Текстовий файл представлено класом `FileRecord`:

```
public class FileRecord
{
    public string FileName { get; set; } = string.Empty;
    // Шлях до файлу у FileService
    public string? StoragePath { get; set; }
    public byte[]? Content { get; set; }
    public string MimeType { get; set; } = "text/plain";
    public long Size { get; set; }
    public DateTime UploadedAt { get; set; } = DateTime.UtcNow;
    [JsonConstructor]
    public FileRecord() { }
    public FileRecord(string fileName, byte[] content, string mimeType =
    ↪  "text/plain")
    {
        FileName = fileName ?? throw new ArgumentNullException(nameof(fileName));
        Content = content ?? throw new ArgumentNullException(nameof(content));
        MimeType = mimeType ?? "text/plain";
        Size = content.Length;
    }
    public FileRecord(string fileName, string storagePath, long size, string
    ↪  mimeType = "text/plain")
    {
        FileName = fileName;
        StoragePath = storagePath;
        Size = size;
        MimeType = mimeType;
    }
}
```

8.2 Завантаження файлу

При виборі файлу через діалог:

1. Файл зчитується у пам'ять
2. Зберігається через `FileService` у папку `uploads/`
3. Створюється `FileRecord` без вмісту файлу
4. Шлях до файлу зберігається у таблиці у відповідному екземплярі класу FileRecord

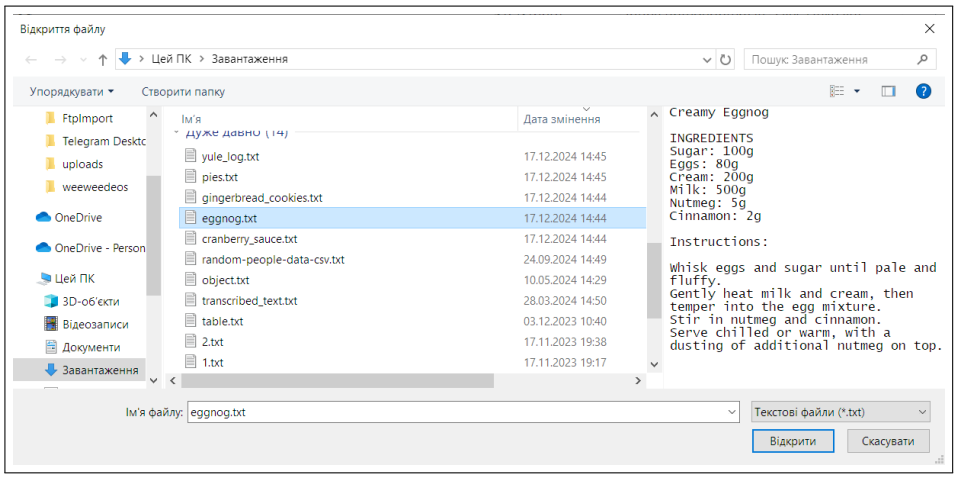


Рис. 13: Вибір текстового файлу

8.3 Перегляд файлу

Після вибору файлу його ім'я відображається як гіперпосилання. Клік по посиланню відкриває файл у Notepad.

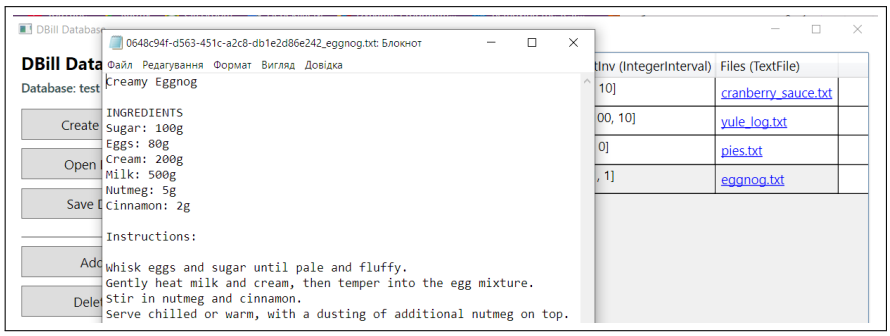


Рис. 14: Відображення завантаженого файлу

Код відкриття файлу з `RowDialog.xaml.cs`:

```
private async Task OpenFile(FileRecord fileRecord)
{
    try
    {
        System.Diagnostics.Process.Start(
            new System.Diagnostics.ProcessStartInfo
        {
            FileName = "notepad.exe",
            Arguments = $"\"{fileRecord.StoragePath}\"",
            UseShellExecute = false
        });
    }
    catch (Exception ex)
    {
        MessageBox.Show(
            $"Error opening the file: {ex.Message}");
    }
}
```

8.4 Управління файлами

`FileService` забезпечує:

- Збереження файлів з унікальними іменами
- Завантаження файлів за шляхом
- Видалення файлів
- Очищення всіх файлів при закритті БД

9 Обробка помилок

9.1 Типи помилок

Система обробляє наступні категорії помилок:

- **Помилки валідації** — некоректні дані від користувача
- **Помилки БД** — порушення цілісності структури
- **Помилки файлової системи** — проблеми доступу до файлів
- **Помилки серіалізації** — проблеми збереження/завантаження

9.2 Валідація даних

Кожен рядок проходить валідацію перед збереженням:

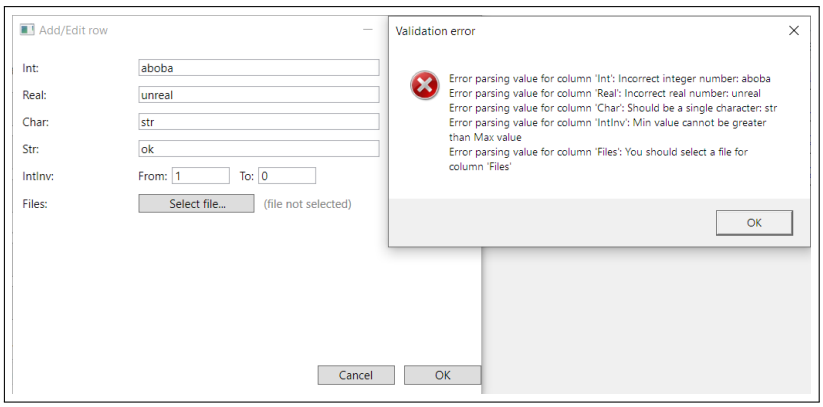


Рис. 15: Приклад повідомлень про помилки валідації

9.3 Спеціалізовані винятки

У проєкті визначено власні класи винятків:

```
public class ValidationException : Exception
{
    public List<string> Errors { get; }

    public ValidationException(List<string> errors) : base("Validation failed")
    {
        Errors = errors;
    }
}
```

```
public class DatabaseException : Exception
{
    public DatabaseException(string message)
        : base(message) { }
}

public class TableNotFoundException : Exception
{
    public TableNotFoundException(string tableName)
        : base($"Table '{tableName}' not found") { }
}
```

9.4 Обробка помилок при завантаженні БД

При завантаженні БД система використовує механізм тимчасового сховища для запобігання пошкодженню поточної БД:

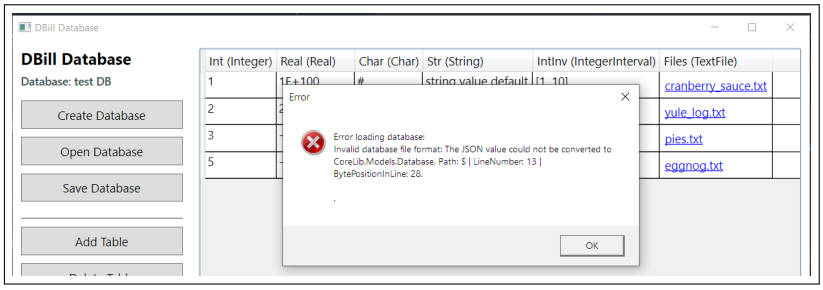


Рис. 16: Повідомлення про помилку при завантаженні БД

Якщо під час завантаження виникає помилка:

- Тимчасові файли автоматично видаляються
- Поточна БД залишається незмінною
- Користувач отримує детальне повідомлення про помилку

9.5 Логування помилок

Критичні помилки відображаються через `MessageBox` з відповідним рівнем важливості:

- `Error` — помилки валідації, проблеми з файлами
- `Information` — інформаційні повідомлення

10 Unit-тестування

10.1 Фреймворк тестування

Для тестування використовується **NUnit**. Тести організовано у три основні класи:

- `DataValidationTests` — тестування валідації даних
- `TableOperationsTests` — тестування операцій над таблицями
- `DatabaseOperationsTests` — тестування операцій над БД

10.2 Тест 1: Валідація цілих чисел

```
[Test]
public void ValidateInteger_WithValidValue_ShouldPass()
{
    // Arrange
    var rowData = new Dictionary<string, object?>
    {
        ["IntCol"] = 42,
        ["RealCol"] = 3.14,
        ["CharCol"] = 'A',
        ["StringCol"] = "test",
        ["IntervalCol"] = new IntegerInterval(1, 10)
    };

    // Act
    var result = _testTable.ValidateRow(rowData);

    // Assert
    Assert.That(result.IsValid, Is.True);
}
```

Мета: перевірити, що валідація пропускає коректні цілочисельні значення.

Результат: тест успішно пройдено, валідація працює правильно для коректних даних.

10.3 Тест 2: Валідація некоректних даних

```
[Test]
public void ValidateInteger_WithInvalidValue_ShouldFail()
{
    // Arrange
    var rowData = new Dictionary<string, object?>
    {
        ["IntCol"] = "not_an_integer",
        ["RealCol"] = 3.14,
        ["CharCol"] = 'A',
        ["StringCol"] = "test",
        ["IntervalCol"] = new IntegerInterval(1, 10)
    };

    // Act
    var result = _testTable.ValidateRow(rowData);

    // Assert
    Assert.That(result.IsValid, Is.False);
    Assert.That(result.Errors.Count > 0, Is.True);
}
```

Мета: перевірити, що валідація відхиляє некоректні значення для типу `integer`.
Результат: тест пройдено, система коректно визначає невідповідність типів.

10.4 Тест 3: Валідація інтервального типу (індивідуальна операція)

```
[Test]
public void ValidateInterval_WithInvalidRange_ShouldFail()
{
    // Arrange
    var rowData = new Dictionary<string, object?>
    {
        ["IntCol"] = 42,
        ["RealCol"] = 3.14,
        ["CharCol"] = 'A',
        ["StringCol"] = "test",
        ["IntervalCol"] = new IntegerInterval(10, 1)
            // Некоректний: Min > Max
```

```
    };

    // Act & Assert
    Assert.Throws<ArgumentException>(() =>
    {
        var result = _testTable.ValidateRow(rowData);
    });
}
```

Мета: перевірити валідацію індивідуального типу `IntegerInterval` — має відхиляти інтервали, де мінімум більший за максимум.

Результат: тест пройдено, система коректно обробляє некоректні інтервали.

10.5 Тест 4: Перейменування колонки (індивідуальна операція)

Цей тест перевіряє індивідуальну операцію згідно варіанту — перейменування колонок таблиці.

```
[Test]
public void RenameColumn_ShouldChangeColumnName()
{
    // Arrange
    var table = _databaseService.GetTable(_testTableName);
    var oldName = "Age";
    var newName = "UserAge";

    // Act
    var result = table?.RenameColumn(oldName, newName);
    var columnNames = table?.GetColumnNames();

    // Assert
    Assert.That(result, Is.True);
    Assert.That(columnNames?.Contains("UserAge"), Is.True);
    Assert.That(columnNames.Contains("Age"), Is.False);
}
```

Мета: перевірити коректність перейменування колонки. Після операції:

- Стара назва "Age"має зникнути зі списку колонок
- Нова назва "UserAge"має з'явитися у списку
- Порядок інших колонок має залишитися незмінним
- Дані у колонці мають зберегтися

Результат: тест пройдено успішно. Операція `RenameColumn()` коректно оновлює метадані таблиці та зберігає цілісність даних.

10.6 Тест 5: Перестановка колонок (індивідуальна операція)

Цей тест перевіряє другу частину індивідуальної операції — зміну порядку колонок у таблиці.

```
[Test]
public void ReorderColumns_ShouldChangeColumnOrder()
{
    // Arrange
    var table = _databaseService.GetTable(_testTableName);
    var newOrder = new List<string>
        { "Salary", "Name", "Age", "ID" };

    // Act
    var result = table?.ReorderColumns(newOrder);
    var columnNames = table?.GetColumnNames();

    // Assert
    Assert.That(result, Is.True);
    Assert.That(newOrder, Is.EqualTo(columnNames));
}
```

Мета: перевірити можливість зміни порядку відображення колонок. Початковий порядок був:

`ID → Name → Age → Salary`

Після перестановки порядок має стати:

`Salary → Name → Age → ID`

Результат: тест пройдено. Метод `ReorderColumns()` успішно змінює порядок колонок без втрати даних.

11 Архітектура та структура проєкту

11.1 Ключові компоненти

11.1.1 Database

Головний клас, що представляє базу даних:

- Містить колекцію таблиць
- Забезпечує операції додавання/видалення таблиць
- Виконує валідацію структури

11.1.2 Table

Клас таблиці з колонками та рядками:

- Зберігає схему (список `Column`)
- Управляє даними (колонкова структура)
- Валідує рядки перед додаванням
- Підтримує операції перейменування колонок

11.1.3 Column

Представляє колонку таблиці:

- Містить ім'я та тип даних
- Зберігає значення у списку
- Підтримує валідацію типів

11.2 Сервісний шар

11.2.1 DatabaseService

Центральний сервіс для роботи з БД:

- Управління поточною базою даних
- Створення, завантаження, збереження БД
- Управління таблицями
- Координація роботи з файлами
- Підтримка тимчасового сховища

11.2.2 TableService

Сервіс для операцій над таблицями:

- Парсинг та валідація рядків
- Операції CRUD над рядками
- Перейменування та перестановка колонок

11.2.3 FileService

Управління файлами:

- Збереження файлів з унікальними іменами
- Завантаження файлів
- Видалення окремих файлів та масове очищення

11.3 Патерни проектування

- **Dependency Injection:** всі сервіси впроваджуються через конструктори, що забезпечує слабкий зв'язок та тестованість.
- **Repository Pattern:** `IDatabaseStorageService` та `IFileStorageService` абстрагують доступ до даних.
- **Strategy Pattern:** різні конвертери для серіалізації складних типів.

12 Особливості реалізації

12.1 Безпечне завантаження БД

Ключова особливість — використання тимчасового сховища при завантаженні:

1. БД завантажується у `tempFiles/`
2. Виконується повна валідація
3. Файли копіюються у `uploads/`
4. Тільки після успішної валідації заміняється поточна БД
5. Тимчасові файли очищаються

Це гарантує, що у разі помилки поточна база залишається незмінною.

12.2 Колонкове зберігання даних

На відміну від традиційного рядкового зберігання, дані організовано колонками:

```
public class Column
{
    public string Name { get; set; }
    public DataType Type { get; set; }
    public List<object?> Values { get; set; }
}
```

Переваги:

- Ефективна валідація типів
- Ефективна реалізація індивідуальної операції (O(n) та O(1) замість O(n*m) та O(m), де n - кількість стовпців, а m - кількість рядків)
- Простіша серіалізація
- Швидкий доступ до всіх значень колонки
- Легші й швидші пошук й обробка файлів

12.3 Валідація на кількох рівнях

1. **Рівень моделі:** конструктори класів перевіряють базові інваріанти
2. **Рівень таблиці:** `ValidateRow()` перевіряє відповідність типам
3. **Рівень БД:** `Validate()` перевіряє цілісність структури
4. **Рівень UI:** попередня валідація при введенні

13 Приклади використання

13.1 Сценарій 1: Створення БД для обліку студентів

1. Створюємо нову БД "University"
2. Створюємо таблицю "Students"з колонками:
   - ID (Integer)
   - Name (String)
   - Age (IntegerInterval) — вікова група
   - Grade (Real)
   - Group (Char)
3. Додаємо рядки з даними студентів
4. Зберігаємо БД на диск

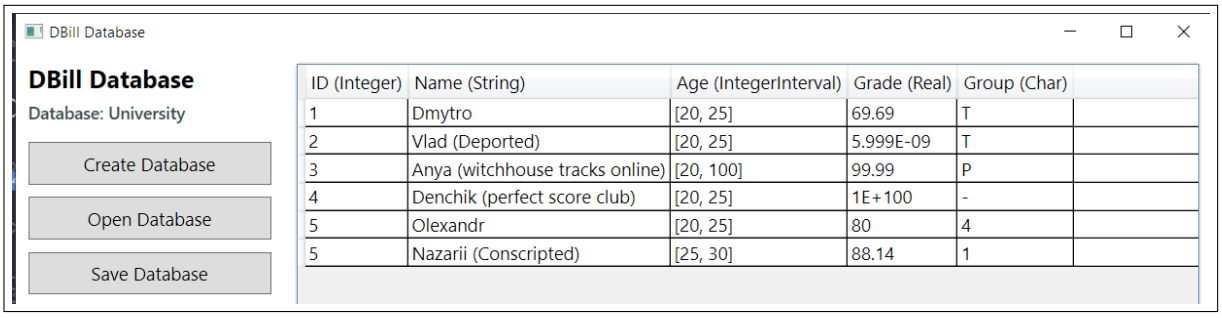


Рис. 17: Приклад таблиці Students

13.2 Сценарій 2: Архів документів

1. Завантажуємо або створюємо нову БД
2. Створюємо таблицю "Documents"
3. Додаємо колонки:
   - DocID (Integer)
   - Title (String)
   - Category (String)
   - Content (TextFile) — текстовий файл документа
4. Завантажуємо текстові файли через діалог
5. Переглядаємо документи кліком по посиланню

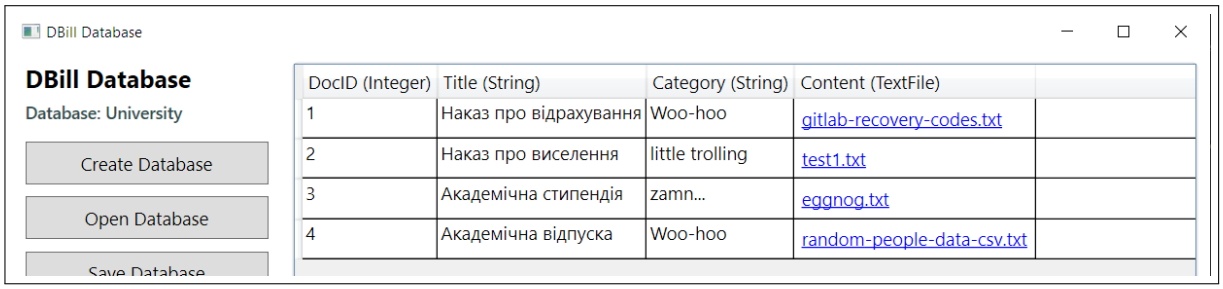


Рис. 18: Приклад таблиці з текстовими файлами

14 Висновок

14.1 Виконані завдання

У рамках етапу 2 успішно реалізовано:

1. **Десктоп-застосунок** на базі WPF з повнофункціональним GUI
2. **Підтримку всіх базових типів:** Integer, Real, Char, String
3. **Додаткові типи згідно варіанту:**
   - Інтервальний тип цілих чисел (IntegerInterval)
   - Текстові файли (TextFile)
4. **Додаткову операцію:** перейменування та перестановка колонок
5. **CRUD операції** над таблицями та рядками
6. **Збереження/завантаження БД** у форматі JSON
7. **Валідацію даних** на всіх рівнях
8. **Обробку помилок** з інформативними повідомленнями
9. **Unit-тестування** з покриттям коду бібліотеки

14.2 Архітектурні переваги

- **Модульність:** розділення на CoreLib, DesktopApp та Tests забезпечує чітке розділення відповідальності.
- **Розширюваність:** завдяки інтерфейсам `IDatabaseStorageService` та `IFileStorageService` легко додати нові способи зберігання.
- **Тестованість:** використання DI та mock-об'єктів дозволяє тестувати компоненти ізольовано.
- **Безпека:** тимчасове сховище при завантаженні запобігає пошкодженню даних.

Додаток А. Код програми

Код програми може бути знайдений за посиланням: https://github.com/8ctag8ne/DBill