КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп’ютерних наук та кібернетики

Звіт

до лабораторної роботи на тему:

ФРАГМЕНТАРНА РЕАЛІЗАЦІЯ СИСТЕМИ УПРАВЛІННЯ ТАБЛИЧНИМИ БАЗАМИ ДАНИХ

Виконав студент 4-го курсу

Групи ТК-41

Клименко Максим Віталійович

Київ – 2023

Постановка задачі

Створити декілька варіантів реалізації системи управління табличними базами даних з використанням запропонованих технологій.

Загальні вимоги:

- кількість таблиць принципово не обмежена;
- кількість полів та записів у кожній таблиці принципово не обмежені;
- забезпечити підтримку типів integer, real, char, string
- реалізувати створення бази;
- реалізувати створення (з валідацією даних) та знищення таблиці з бази;
- реалізувати перегляд та редагування рядків таблиці;
- реалізувати збереження табличної бази на диску та, навпаки, її зчитування з диску.

Додаткові типи та операція над таблицями – варіант 26:

2) html-файли; stringInvl;

6)прямий добуток двох таблиць

Для розробки використовував мову програмування c++ та фреймворк qt для GUI.

Розробка локальної версії СУТБД

• Розробка класів

Database – клас бази даних.

```
class database
{
public:
    database(const QString& name);

    QString get_name();
    std::vector<table> get_tables();
    void add_table(const table& table);

    void remove_table(const int& index){
        tables__.erase(tables__.begin()+index);
    }
    table& get_table(const int index){
        return tables__[index];
    }
private:
    QString name__;
    std::vector<table> tables__;
};
```

Table — клас таблиць.

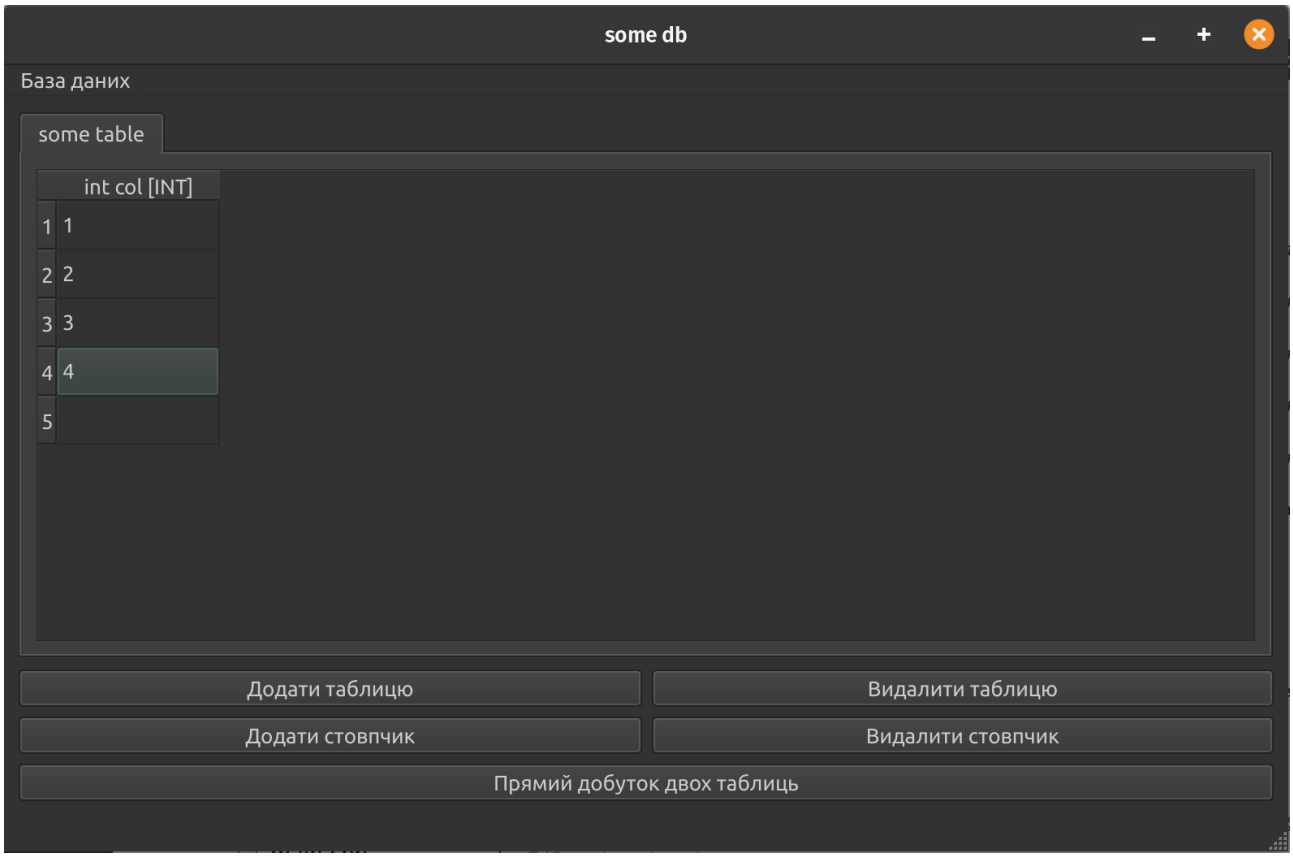
```
class table
{
public:
    table(const QString& name);


    std::shared_ptr<column> get_column(int index){
        return columns__[index];
    }

    void add_column(std::shared_ptr<column> column);

    void remove_column(int index){
        columns__.erase(columns__.begin() + index);
    }

    std::vector<std::shared_ptr<column>> get_columns();
private:
    QString name__;
    std::vector<std::shared_ptr<column>> columns__;
};
```

Column — абстрактний клас стовпчиків. Слугує для наслідування типізованих стовпчиків.

```
enum TYPE{
    INT = 0,
    REAL,
    CHAR,
    STRING,
    HTML,
    STRINGINVL
};


class column
{
public:
    column(const QString& name);
    virtual bool validate(const std::string& value) = 0;
    TYPE get_type();
protected:
    TYPE type_;

private:
    QString name__;

};
    class column_char : public column
    {
    public:
        column_char(const QString& name);

        bool validate(const std::string& value);
    };
  class column_html : public column
  {
  public:
      column_html(const QString& name);
      bool validate(const std::string& value);
  private:
      bool fileExists(const std::string& fileName);
  };
    class column_int : public column
    {
    public:
        column_int(const QString& name);;
        bool validate(const std::string& value);
    };
     class column_real : public column
     {
     public:
         column_real(const QString& name);;
         bool validate(const std::string& value);
     };
    class column_string : public column
    {
    public:
        column_string(const QString& name);

        bool validate(const std::string& value);
    };
     class column_string_interval : public column
     {
     public:
         column_string_interval(const QString& name);;
         bool validate(const std::string& value);

     private:
         void remove_spaces(std::string& val);
     };
```

Де клас Row ?!

Нам не потрібно зберігати рядки самим. Вони зберігаються самі в класі QTableWidget об’єкті(цей тип із qt). Чесно кажучи, стовпчики теж можна було б не зберігати, якби не валідація.



По суті, база даних — сукупність tabs (вкладок), які в свою чергу містять об’єкти QTableWidget.

• Тестування

Для тестування використовував засоби фреймворку qt, а саме — QtTests

Зробив необхідних “3+” тестів. Один на перевірку добутку таблиць. Інші 2 - на перевірку валідації типів int та real.

```
class DBMS : public QObject
{
    Q_OBJECT

public:


private slots:
    void test_int_column();
    void test_real_column();
    void test_table_product();

};
```

```
void DBMS::test_int_column()
{
    column_int test("");

    bool test1 = test.validate("12");
    bool test2 = test.validate("12.5");
    bool test3 = test.validate("123a");

    QCOMPARE(test1, true);
    QCOMPARE(test2, false);
    QCOMPARE(test3, false);
}

void DBMS::test_real_column(){
    column_real test("");

    bool test1 = test.validate("12");
    bool test2 = test.validate("12.5");
    bool test3 = test.validate("123a.5");
    bool test4 = test.validate("12,5");

    QCOMPARE(test1, true);
    QCOMPARE(test2, true);
    QCOMPARE(test3, false);
    QCOMPARE(test4, false);

}
void DBMS::test_table_product(){


    QTableWidget leftTable;
    leftTable.setRowCount(2);
    leftTable.setColumnCount(2);
    leftTable.setItem(0, 0, new QTableWidgetItem("A"));
    leftTable.setItem(0, 1, new QTableWidgetItem("B"));
    leftTable.setItem(1, 0, new QTableWidgetItem("C"));
    leftTable.setItem(1, 1, new QTableWidgetItem("D"));


    QTableWidget rightTable;
    rightTable.setRowCount(2);
    rightTable.setColumnCount(1);
    rightTable.setItem(0, 0, new QTableWidgetItem("1"));
    rightTable.setItem(1, 0, new QTableWidgetItem("2"));


    QTableWidget resultTable;
    cartesian_product(&leftTable, &rightTable, &resultTable);


    QCOMPARE(resultTable.rowCount(), 4);
    QCOMPARE(resultTable.columnCount(), 3);
    QCOMPARE(resultTable.item(0, 0)->text(), QString("A"));
    QCOMPARE(resultTable.item(0, 1)->text(), QString("B"));
    QCOMPARE(resultTable.item(0, 2)->text(), QString("1"));



    for (int i = 0; i < resultTable.rowCount(); ++i) {
        for (int j = 0; j < resultTable.columnCount(); ++j) {
            delete resultTable.item(i, j);
        }
    }
}
```

```cpp
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    DBMS dbms;

    int result = QTest::qExec(&dbms, argc, argv);

    return result;
}

#include "tst_dbms.moc"
```

```
13:30:47: Запуск /home/maksym/uni4/IT/DBMS/build-Tests-Desktop_Qt_6_6_0_GCC_64bit-Debug/DBMS...
********* Start testing of DBMS *********
Config: Using QtTest library 6.6.0, Qt 6.6.0 (x86_64-little_endian-lp64 shared (dynamic) release build; by GCC 10.3.1 20210422 (Red Hat 10.3.1-1)),
pop 22.04
PASS   : DBMS::initTestCase()
PASS   : DBMS::test_int_column()
PASS   : DBMS::test_real_column()
PASS   : DBMS::test_table_product()
PASS   : DBMS::cleanupTestCase()
Totals: 5 passed, 0 failed, 0 skipped, 0 blacklisted, 14ms
********* Finished testing of DBMS *********
13:30:47: /home/maksym/uni4/IT/DBMS/build-Tests-Desktop_Qt_6_6_0_GCC_64bit-Debug/DBMS завершився з кодом 0
```

• Забезпечення інтерфейсу користувача на основі форм

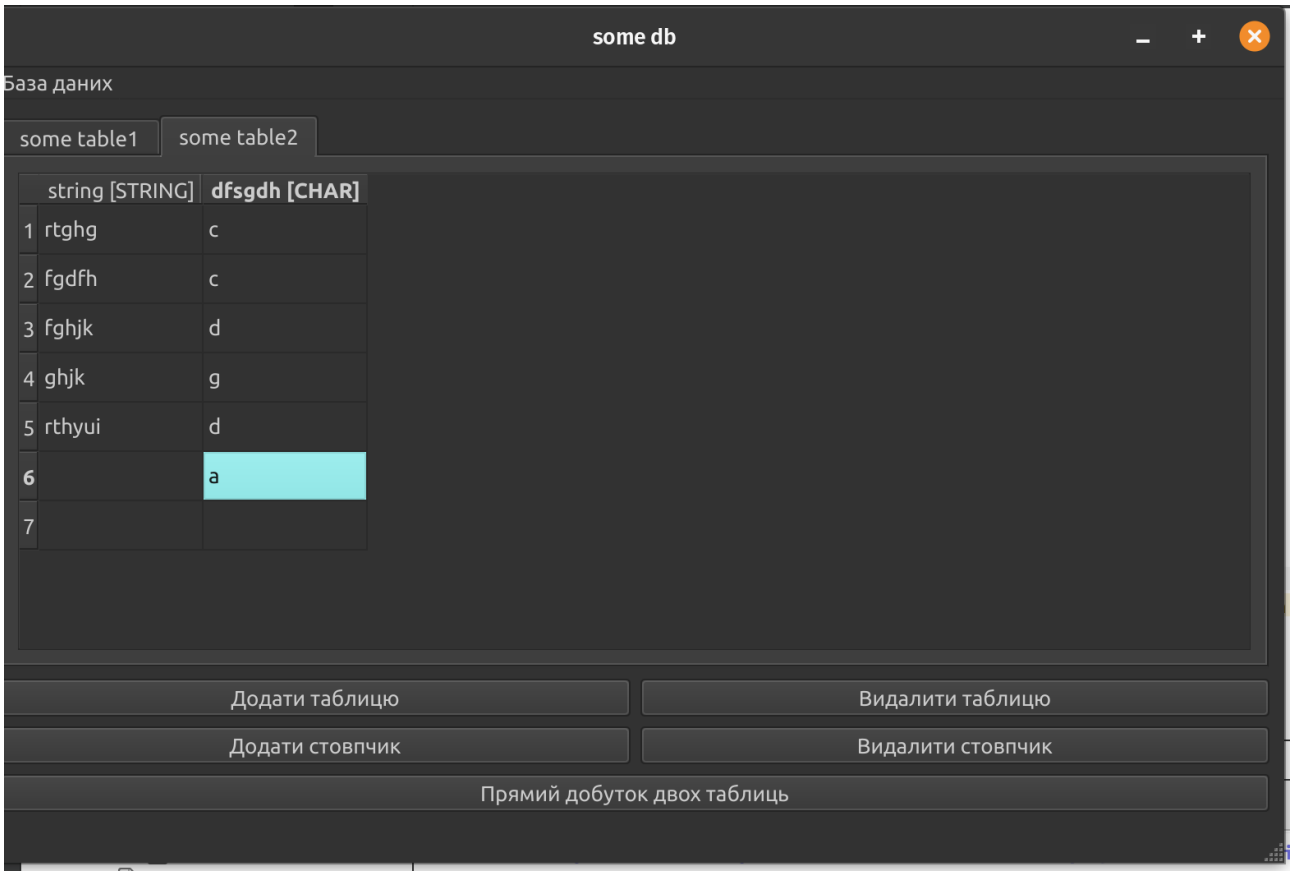
Як було написано вище, використав qt framework.:

- Додати базу даних – створити нову базу даних.
- Додати таблицю – додати до створеної бази даних таблицю. Якщо таблиця з таким іменем вже створена, видається повідомлення про помилку.
- Додати стовпчик – додати до таблиці стовпчик. Якщо стовпчик з таким іменем вже присутній у цій таблиці, видається повідомлення про помилку.
- Додати рядок – додати рядок до таблиці.
- Видалити базу даних – видалити поточну базу даних.
- Видалити таблицю – видалити таблицю.
- Видалити стовпчик – видалити стовпчик за індексом.
- Видалити рядок – видалення рядку відбувається шляхом занулення всіх клітин рядка.

- Прямий добуток – отримати прямий добуток двох таблиць за їх індексами.
- Відкрити – відкрити базу даних з диску (json-файл).
- Зберегти – база даних записується на диск (json-файл).

При вставленні рядків таблиці відбувається валідація і, якщо тип введеного значення не співпадає з типом стовпчика, видається повідомлення про помилку.

Нижче подано приклад функціонування застосунку:



Більш детально з кодом можна ознайомитись у моєму [репозиторію](#).