Структура папки Console/

```
Console/
  ├──── contact.h
  ├──── contact.cpp
  └──── main.cpp
```

contact.h

```
#ifndef CONTACT_H
#define CONTACT_H

#include <string>
#include <vector>
#include <regex>

struct Phone {
    std::string label;  // Тип номера (домашний, рабочий, мобильный)
    std::string number; // Сам номер
};

class Contact {
public:
    std::string firstName, lastName, middleName;
    std::string address;
    std::string birthDate; // YYYY-MM-DD
    std::string email;
    std::vector<Phone> phones;

    Contact() = default;

    std::string serialize() const;             // сохранить в строку
    static Contact deserialize(const std::string &line); // загрузить из строки

    // Проверки корректности данных
    static bool validateName(const std::string &s);
    static bool validatePhone(const std::string &s);
    static bool validateDate(const std::string &s);
    static bool validateEmail(const std::string &s);
};

#endif
```

contact.cpp

```
#include "contact.h"
#include <sstream>
#include <iomanip>
#include <ctime>
#include <iostream>

static std::string escape(const std::string &s) {
    std::string out = s;
    for (char &c : out)
        if (c == '\n' || c == '\r') c = ' ';
    return out;
}

std::string Contact::serialize() const {
    std::ostringstream ss;
    ss << escape(lastName) << "," << escape(firstName) << "," <<
escape(middleName) << ",";
    ss << escape(address) << "," << escape(birthDate) << "," <<
escape(email) << ",";
    bool first = true;
    for (auto &p : phones) {
        if (!first) ss << ";";
        ss << p.label << "|" << p.number;
        first = false;
    }
    return ss.str();
}

Contact Contact::deserialize(const std::string &line) {
    Contact c;
    std::istringstream ss(line);
    std::getline(ss, c.lastName, ',');
    std::getline(ss, c.firstName, ',');
    std::getline(ss, c.middleName, ',');
    std::getline(ss, c.address, ',');
    std::getline(ss, c.birthDate, ',');
    std::getline(ss, c.email, ',');
    std::string phones;
    std::getline(ss, phones, ',');
    if (!phones.empty()) {
        std::istringstream ps(phones);
        std::string part;
        while (std::getline(ps, part, ';')) {
```

```
            size_t pos = part.find('|');
            if (pos != std::string::npos) {
                Phone p{part.substr(0, pos), part.substr(pos + 1)};
                c.phones.push_back(p);
            }
        }
    }
    return c;
}

bool Contact::validateName(const std::string &s) {
    if (s.empty()) return false;
    if (s.front() == '-' || s.back() == '-') return false;
    for (char c : s) {
        if (!(isalpha((unsigned char)c) || isdigit((unsigned char)c) ||
c == '-' || c == ' '))
            return false;
    }
    return true;
}

bool Contact::validatePhone(const std::string &s) {
    std::string digits;
    for (char c : s)
        if (isdigit((unsigned char)c)) digits.push_back(c);
    return digits.size() >= 7 && digits.size() <= 15;
}

bool Contact::validateDate(const std::string &s) {
    std::regex rx(R"(^\s*(\d{4})-(\d{2})-(\d{2})\s*$)");
    std::smatch m;
    if (!std::regex_match(s, m, rx)) return false;
    int y = stoi(m[1]), mo = stoi(m[2]), d = stoi(m[3]);
    if (mo < 1 || mo > 12) return false;
    int mdays[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    bool leap = ((y % 4 == 0 && y % 100 != 0) || (y % 400 == 0));
    if (leap) mdays[2] = 29;
    if (d < 1 || d > mdays[mo]) return false;
    return true;
}

bool Contact::validateEmail(const std::string &s) {
    std::regex
rx(R"(^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$)");
```

```
    return std::regex_match(s, rx);
}

main.cpp
#include "contact.h"
#include <iostream>
#include <vector>
#include <fstream>

using namespace std;

void save(const string &fname, const vector<Contact> &list) {
    ofstream f(fname);
    for (auto &c : list) f << c.serialize() << "\n";
    cout << "Saved " << list.size() << " contacts to " << fname << endl;
}

vector<Contact> load(const string &fname) {
    vector<Contact> out;
    ifstream f(fname);
    string line;
    while (getline(f, line)) {
        if (!line.empty()) out.push_back(Contact::deserialize(line));
    }
    cout << "Loaded " << out.size() << " contacts from " << fname << endl;
    return out;
}

int main() {
    vector<Contact> contacts;
    string fname = "contacts.txt";
    cout << "PhoneBook Console\nCommands: add, list, save, load, exit\n";
    while (true) {
        cout << "> ";
        string cmd;
        if (!(cin >> cmd)) break;
        if (cmd == "add") {
            Contact c;
            cin.ignore();
            cout << "Last name: "; getline(cin, c.lastName);
            cout << "First name: "; getline(cin, c.firstName);
            cout << "Middle name: "; getline(cin, c.middleName);
            cout << "Address: "; getline(cin, c.address);
            cout << "Birth YYYY-MM-DD: "; getline(cin, c.birthDate);
```

```
            cout << "Email: "; getline(cin, c.email);
            int n;
            cout << "How many phones? "; cin >> n; cin.ignore();
            for (int i = 0; i < n; i++) {
                Phone p;
                cout << "Label: "; getline(cin, p.label);
                cout << "Number: "; getline(cin, p.number);
                c.phones.push_back(p);
            }
            if (!Contact::validateName(c.firstName)
|| !Contact::validateEmail(c.email)) {
                cout << "Invalid input!\n";
                continue;
            }
            contacts.push_back(c);
            cout << "Added.\n";
        } else if (cmd == "list") {
            for (size_t i = 0; i < contacts.size(); ++i) {
                cout << i + 1 << ". " << contacts[i].lastName << " " <<
contacts[i].firstName
                     << " - " << contacts[i].email << endl;
            }
        } else if (cmd == "save") {
            save(fname, contacts);
        } else if (cmd == "load") {
            contacts = load(fname);
        } else if (cmd == "exit") break;
        else cout << "Unknown command\n";
    }
}
```

Как запустить

1- Скопируй эти три файла в папку `Console/`.
2- Скомпилируй в консоли:

g++ -std=c++17 main.cpp contact.cpp -o phonebook

3- Запусти:

./phonebook

4- Используй команды:

- `add` — добавить контакт
- `list` — показать список
- `save` — сохранить в файл
- `load` — загрузить из файла
- `exit` — выход

Структура папки QtApp/

```
QtApp/
  ├── PhoneBook.pro
  ├── main.cpp
  ├── mainwindow.h
  ├── mainwindow.cpp
  ├── ui_mainwindow.h
  ├── addeditdialog.h
  ├── addeditdialog.cpp
  ├── ui_addeditdialog.h
  ├── contactmodel.h
  ├── contactmodel.cpp
  ├── dbmanager.h
  ├── dbmanager.cpp
```

PhoneBook.pro

```
QT += core gui sql
CONFIG += c++17
```

```
TARGET = PhoneBook

TEMPLATE = app


SOURCES += main.cpp \

    mainwindow.cpp \

    addeditdialog.cpp \

    contactmodel.cpp \

    dbmanager.cpp


HEADERS += mainwindow.h \

    addeditdialog.h \

    contactmodel.h \

    dbmanager.h \

    ui_mainwindow.h \

    ui_addeditdialog.h
```

main.cpp

```
#include <QApplication>

#include "mainwindow.h"


int main(int argc, char *argv[]) {

    QApplication a(argc, argv);

    MainWindow w;

    w.show();

    return a.exec();
```

```
}
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QStandardItemModel>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_add();
    void on_edit();
    void on_delete();
    void on_search();
    void on_connectDb();
```

```
    void on_syncToDb();


private:
    Ui::MainWindow *ui;
    QStandardItemModel *model;
    void loadFromFile(const QString &fname);
    void saveToFile(const QString &fname);
};
#endif
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "addeditdialog.h"
#include "dbmanager.h"

#include <QFile>
#include <QTextStream>
#include <QMessageBox>
#include <QInputDialog>
#include <QSqlQuery>
#include <QSqlError>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow) {
```

```
        ui->setupUi(this);



        model = new QStandardItemModel(this);

        model->setHorizontalHeaderLabels({"Last", "First",
"Email"});

        ui->tableView->setModel(model);



        loadFromFile("contacts_qt.db");



        connect(ui->addButton, &QPushButton::clicked, this,
&MainWindow::on_add);

        connect(ui->editButton, &QPushButton::clicked, this,
&MainWindow::on_edit);

        connect(ui->deleteButton, &QPushButton::clicked, this,
&MainWindow::on_delete);

        connect(ui->searchButton, &QPushButton::clicked, this,
&MainWindow::on_search);

        connect(ui->saveButton, &QPushButton::clicked, this, [this]()
{ saveToFile("contacts_qt.db"); });

        connect(ui->dbConnectButton, &QPushButton::clicked, this,
&MainWindow::on_connectDb);

        connect(ui->syncButton, &QPushButton::clicked, this,
&MainWindow::on_syncToDb);

    }



    MainWindow::~MainWindow() { delete ui; }
```

```
void MainWindow::loadFromFile(const QString &fname) {
    QFile f(fname);
    if (!f.open(QIODevice::ReadOnly | QIODevice::Text)) return;
    QTextStream in(&f);
    while (!in.atEnd()) {
        auto line = in.readLine().split(',');
        if (line.size() >= 3) {
            QList<QStandardItem *> row;
            row << new QStandardItem(line[0]) << new
QStandardItem(line[1]) << new QStandardItem(line[2]);
            model->appendRow(row);
        }
    }
}

void MainWindow::saveToFile(const QString &fname) {
    QFile f(fname);
    if (!f.open(QIODevice::WriteOnly | QIODevice::Text)) return;
    QTextStream out(&f);
    for (int r = 0; r < model->rowCount(); ++r)
        out << model->item(r, 0)->text() << "," << model->item(r,
1)->text() << "," << model->item(r, 2)->text() << "\n";
}

void MainWindow::on_add() {
```

```
        AddEditDialog dlg(this);
        if (dlg.exec() == QDialog::Accepted) {
            auto r = dlg.record();
            QList<QStandardItem *> row;
            row << new QStandardItem(r.last) << new
QStandardItem(r.first) << new QStandardItem(r.email);
            model->appendRow(row);
        }
    }

    void MainWindow::on_edit() {
        auto sel = ui->tableView->selectionModel()->selectedRows();
        if (sel.isEmpty()) return;
        int r = sel.first().row();
        AddEditDialog dlg(this);
        dlg.setRecord({model->item(r,0)->text(),
model->item(r,1)->text(), "", "", "", model->item(r,2)->text(), {}});
        if (dlg.exec() == QDialog::Accepted) {
            auto nr = dlg.record();
            model->item(r, 0)->setText(nr.last);
            model->item(r, 1)->setText(nr.first);
            model->item(r, 2)->setText(nr.email);
        }
    }
```

```
void MainWindow::on_delete() {
    auto sel = ui->tableView->selectionModel()->selectedRows();
    if (!sel.isEmpty()) model->removeRow(sel.first().row());
}

void MainWindow::on_search() {
    bool ok;
    QString term = QInputDialog::getText(this, "Search", "Enter text:", QLineEdit::Normal, "", &ok);
    if (!ok || term.isEmpty()) return;
    for (int r = 0; r < model->rowCount(); ++r)
        for (int c = 0; c < model->columnCount(); ++c)
            if (model->item(r, c)->text().contains(term, Qt::CaseInsensitive)) {
                ui->tableView->selectRow(r);
                return;
            }
    QMessageBox::information(this, "Search", "No matches found.");
}

void MainWindow::on_connectDb() {
    bool ok;
    QString host = QInputDialog::getText(this, "Host", "Host:", QLineEdit::Normal, "localhost", &ok);
```

```
        if (!ok) return;

        int port = QInputDialog::getInt(this, "Port", "Port:", 5432,
1, 65535, 1, &ok);

        if (!ok) return;

        QString db = QInputDialog::getText(this, "DB", "Database:",
QLineEdit::Normal, "phonebook", &ok);

        if (!ok) return;

        QString user = QInputDialog::getText(this, "User", "User:",
QLineEdit::Normal, "postgres", &ok);

        if (!ok) return;

        QString pass = QInputDialog::getText(this, "Password",
"Password:", QLineEdit::Password, "", &ok);

        if (!ok) return;

        DBManager::connectToPostgres(host, port, db, user, pass);

    }



    void MainWindow::on_syncToDb() {

        auto db = DBManager::db();

        if (!db.isOpen()) {

            QMessageBox::warning(this, "DB", "Not connected to
database!");

            return;

        }

        QSqlQuery q(db);

        q.exec("CREATE TABLE IF NOT EXISTS contacts (id serial primary
key, last text, first text, email text)");
```

```
        q.exec("TRUNCATE contacts");

        q.prepare("INSERT INTO contacts(last, first, email)
VALUES(:l, :f, :e)");

        for (int r = 0; r < model->rowCount(); ++r) {

            q.bindValue(":l", model->item(r, 0)->text());

            q.bindValue(":f", model->item(r, 1)->text());

            q.bindValue(":e", model->item(r, 2)->text());

            q.exec();

        }

        QMessageBox::information(this, "DB", "Synced to
PostgreSQL!");

    }
```

ui_mainwindow.h

(создаёт основное окно с таблицей и кнопками)

```
    #ifndef UI_MAINWINDOW_H

    #define UI_MAINWINDOW_H



    #include <QtWidgets>



    namespace Ui {

    class MainWindow {

    public:

        QWidget *centralWidget;

        QTableView *tableView;
```

```
        QPushButton *addButton, *editButton, *deleteButton;
        QPushButton *searchButton, *saveButton, *dbConnectButton,
*syncButton;

        void setupUi(QMainWindow *MainWindow) {
            MainWindow->setWindowTitle("PhoneBook - Qt");
            centralWidget = new QWidget(MainWindow);
            MainWindow->setCentralWidget(centralWidget);

            QVBoxLayout *vLayout = new QVBoxLayout(centralWidget);
            QHBoxLayout *hLayout = new QHBoxLayout();

            addButton = new QPushButton("Add");
            editButton = new QPushButton("Edit");
            deleteButton = new QPushButton("Delete");
            searchButton = new QPushButton("Search");
            saveButton = new QPushButton("Save");
            dbConnectButton = new QPushButton("Connect DB");
            syncButton = new QPushButton("Sync -> DB");

            hLayout->addWidget(addButton);
            hLayout->addWidget(editButton);
            hLayout->addWidget(deleteButton);
            hLayout->addWidget(searchButton);
```

```
            hLayout->addWidget(saveButton);
            hLayout->addWidget(dbConnectButton);
            hLayout->addWidget(syncButton);

            vLayout->addLayout(hLayout);

            tableView = new QTableView();
            vLayout->addWidget(tableView);

            MainWindow->resize(900, 600);
        }
    };
    }
    #endif
```

addeditdialog.h

(описание диалога добавления / редактирования контакта)

```
#ifndef ADDEDITDIALOG_H
#define ADDEDITDIALOG_H

#include <QDialog>

namespace Ui { class AddEditDialog; }

struct Phone {
```

```
    QString label;
    QString number;
};

struct ContactRecord {
    QString last, first, middle, address, birth, email;
    QList<Phone> phones;
};

class AddEditDialog : public QDialog {
    Q_OBJECT
public:
    explicit AddEditDialog(QWidget *parent = nullptr);
    ~AddEditDialog();

    void setRecord(const ContactRecord &r);
    ContactRecord record() const;

private slots:
    void on_addPhoneButton_clicked();
    void on_removePhoneButton_clicked();

private:
    Ui::AddEditDialog *ui;
};
```

#endif

addeditdialog.cpp

(логика диалога — ввод, добавление/удаление телефонов, возврат данных)

```
#include "addeditdialog.h"
#include "ui_addeditdialog.h"
#include <QMessageBox>

AddEditDialog::AddEditDialog(QWidget *parent)
    : QDialog(parent), ui(new Ui::AddEditDialog) {
    ui->setupUi(this);
}

AddEditDialog::~AddEditDialog() { delete ui; }

void AddEditDialog::setRecord(const ContactRecord &r) {
    ui->lastLine->setText(r.last);
    ui->firstLine->setText(r.first);
    ui->middleLine->setText(r.middle);
    ui->addressLine->setText(r.address);
    ui->birthLine->setText(r.birth);
    ui->emailLine->setText(r.email);
    ui->phonesList->clear();
```

```
    for (auto &p : r.phones)
        ui->phonesList->addItem(p.label + ": " + p.number);
}

ContactRecord AddEditDialog::record() const {
    ContactRecord r;
    r.last = ui->lastLine->text();
    r.first = ui->firstLine->text();
    r.middle = ui->middleLine->text();
    r.address = ui->addressLine->text();
    r.birth = ui->birthLine->text();
    r.email = ui->emailLine->text();
    for (int i = 0; i < ui->phonesList->count(); ++i) {
        QString text = ui->phonesList->item(i)->text();
        int pos = text.indexOf(':');
        Phone p;
        if (pos > 0) {
            p.label = text.left(pos).trimmed();
            p.number = text.mid(pos + 1).trimmed();
        } else {
            p.label = "phone";
            p.number = text;
        }
        r.phones.append(p);
```

```
    }

    return r;

}


void AddEditDialog::on_addPhoneButton_clicked() {

    QString lbl = ui->phoneLabel->text().trimmed();

    QString num = ui->phoneNumber->text().trimmed();

    if (lbl.isEmpty() || num.isEmpty()) {

        QMessageBox::warning(this, "Warning", "Enter label and
number");

        return;

    }

    ui->phonesList->addItem(lbl + ": " + num);

    ui->phoneLabel->clear();

    ui->phoneNumber->clear();

}


void AddEditDialog::on_removePhoneButton_clicked() {

    auto items = ui->phonesList->selectedItems();

    for (auto it : items) delete it;

}
```

ui_addeditdialog.h

(визуальная часть диалога — поля ввода, список телефонов, кнопки)

```
#ifndef UI_ADDEDITDIALOG_H
```

```
#define UI_ADDEDITDIALOG_H


#include <QtWidgets>


namespace Ui {

class AddEditDialog {

public:

    QWidget *dialog;

    QLineEdit *lastLine, *firstLine, *middleLine, *addressLine,
*birthLine, *emailLine;

    QListWidget *phonesList;

    QLineEdit *phoneLabel, *phoneNumber;

    QPushButton *addPhoneButton, *removePhoneButton;

    QPushButton *okButton, *cancelButton;


    void setupUi(QDialog *parent) {

        parent->setWindowTitle("Add / Edit Contact");

        QVBoxLayout *mainLayout = new QVBoxLayout(parent);


        QFormLayout *form = new QFormLayout();

        lastLine = new QLineEdit();

        firstLine = new QLineEdit();

        middleLine = new QLineEdit();

        addressLine = new QLineEdit();
```

```
birthLine = new QLineEdit();
emailLine = new QLineEdit();

form->addRow("Last name:", lastLine);
form->addRow("First name:", firstLine);
form->addRow("Middle name:", middleLine);
form->addRow("Address:", addressLine);
form->addRow("Birth (YYYY-MM-DD):", birthLine);
form->addRow("Email:", emailLine);
mainLayout->addLayout(form);

mainLayout->addWidget(new QLabel("Phones:"));
phonesList = new QListWidget();
mainLayout->addWidget(phonesList);

QHBoxLayout *phoneLayout = new QHBoxLayout();
phoneLabel = new QLineEdit();
phoneNumber = new QLineEdit();
addPhoneButton = new QPushButton("Add phone");
removePhoneButton = new QPushButton("Remove selected");
phoneLayout->addWidget(phoneLabel);
phoneLayout->addWidget(phoneNumber);
phoneLayout->addWidget(addPhoneButton);
phoneLayout->addWidget(removePhoneButton);
```

```
        mainLayout->addLayout(phoneLayout);



        QHBoxLayout *buttons = new QHBoxLayout();

        okButton = new QPushButton("OK");

        cancelButton = new QPushButton("Cancel");

        buttons->addStretch();

        buttons->addWidget(okButton);

        buttons->addWidget(cancelButton);

        mainLayout->addLayout(buttons);



        QObject::connect(addPhoneButton, &QPushButton::clicked, parent,
[parent]() {

            QMetaObject::invokeMethod(parent,
"on_addPhoneButton_clicked");

        });

        QObject::connect(removePhoneButton, &QPushButton::clicked,
parent, [parent]() {

            QMetaObject::invokeMethod(parent,
"on_removePhoneButton_clicked");

        });

        QObject::connect(okButton, &QPushButton::clicked, parent,
&QDialog::accept);

        QObject::connect(cancelButton, &QPushButton::clicked, parent,
&QDialog::reject);

    }

};
```

```
}

#endif
```

Структура папки DB/

```
DB/
  ├── dbmanager.h
  ├── dbmanager.cpp
  ├── migrate.sql
  └── README_DB.txt
```

dbmanager.h

Класс для подключения к PostgreSQL через драйвер Qt `QPSQL`

```
#ifndef DBMANAGER_H
#define DBMANAGER_H

#include <QString>
#include <QSqlDatabase>

class DBManager {
public:
    // Подключение к PostgreSQL
    static bool connectToPostgres(const QString &host,
                                  int port,
                                  const QString &db,
                                  const QString &user,
                                  const QString &pass);

    // Получить текущее подключение
```

```
    static QSqlDatabase db();

    // Закрыть соединение
    static void close();

private:
    static QSqlDatabase database;
};

#endif
```

dbmanager.cpp

```
#include "dbmanager.h"
#include <QSqlError>
#include <QDebug>

QSqlDatabase DBManager::database;

bool DBManager::connectToPostgres(const QString &host, int port, const
QString &db, const QString &user, const QString &pass) {
    database = QSqlDatabase::addDatabase("QPSQL");
    database.setHostName(host);
    database.setPort(port);
    database.setDatabaseName(db);
    database.setUserName(user);
    database.setPassword(pass);
```

```
    if (!database.open()) {
        qWarning() << "❌ Не удалось подключиться к PostgreSQL:" << database.lastError().text();
        return false;
    }

    qDebug() << "✅ Подключение к PostgreSQL успешно.";
    return true;
}

QSqlDatabase DBManager::db() { return database; }

void DBManager::close() {
    if (database.isOpen())
        database.close();
}
```

migrate.sql

SQL-скрипт для создания таблиц в PostgreSQL.
Выполни эти команды в pgAdmin или `psql` один раз перед запуском Qt-приложения.

```
-- Создание таблицы контактов

CREATE TABLE IF NOT EXISTS contacts (
    id SERIAL PRIMARY KEY,
    last TEXT,
```

```
    first TEXT,
    middle TEXT,
    address TEXT,
    birth DATE,
    email TEXT
);

-- Создание таблицы телефонов (много номеров у одного контакта)
CREATE TABLE IF NOT EXISTS phones (
    id SERIAL PRIMARY KEY,
    contact_id INTEGER REFERENCES contacts(id) ON DELETE CASCADE,
    label TEXT,
    number TEXT
);
```

README_DB.txt

```
   PostgreSQL Integration Guide
------------------------------

1. Установи PostgreSQL:
   Windows: https://www.postgresql.org/download/
   Ubuntu:  sudo apt install postgresql libpq-dev
            sudo apt install libqt5sql5-psql  (или Qt6 аналог)

2. Создай базу данных:
```

createdb phonebook

3. Выполни SQL-скрипт:

psql -d phonebook -f migrate.sql

4. В Qt-приложении нажми кнопку "Connect DB" и введи:

Host: localhost

Port: 5432

DB: phonebook

User: postgres

Password: <твой пароль>

5. Кнопка "Sync -> DB" синхронизирует текущие данные из таблицы GUI в PostgreSQL.

Как это работает в QtApp:

- Когда ты нажимаешь **"Connect DB"**, вызывается `DBManager::connectToPostgres()`.

- При нажатии **"Sync → DB"** в `mainwindow.cpp` приложение:

- Проверяет, открыто ли соединение;

- Создаёт таблицу `contacts`, если её нет;

- Очищает таблицу (для простоты);

- Записывает все строки из таблицы Qt (`QTableView`) в PostgreSQL

Папка Advanced/

Advanced/

├── contact_advanced.h

├── contact_advanced.cpp

└── main.cpp ← (необязательно, но добавим для демонстрации)

contact_advanced.h

```
#ifndef CONTACT_ADVANCED_H

#define CONTACT_ADVANCED_H


#include <string>

#include <atomic>

#include <new>

#include <iostream>


class ContactAdvanced {

public:

    std::string first, last, email;


    // ✅ Счётчики для анализа

    static std::atomic<int> news;

    static std::atomic<int> constructed;

    static std::atomic<int> copied;

    static std::atomic<int> moved;

    static std::atomic<int> destroyed;
```

```
// --- К о н с т р у к т о р ы ---
ContactAdvanced() { ++constructed; }

ContactAdvanced(const ContactAdvanced &other)
    : first(other.first), last(other.last), email(other.email) {
    ++copied;
}

ContactAdvanced(ContactAdvanced &&other) noexcept
    : first(std::move(other.first)),
      last(std::move(other.last)),
      email(std::move(other.email)) {
    ++moved;
}

~ContactAdvanced() { ++destroyed; }

// --- П е р е о п р е д е л я е м  о п е р а т о р  new ---
static void* operator new(std::size_t sz) {
    ++news;
    return ::operator new(sz);
}

static void operator delete(void *p) noexcept {
```

```
        ::operator delete(p);
    }

    // --- Методы для статистики ---
    static void resetCounters() {
        news = 0; constructed = 0; copied = 0; moved = 0; destroyed = 0;
    }

    static void printCounters() {
        std::cout << "Operator new calls: " << news.load()
                  << "\nConstructed: " << constructed.load()
                  << "\nCopied: " << copied.load()
                  << "\nMoved: " << moved.load()
                  << "\nDestroyed: " << destroyed.load()
                  << std::endl;
    }
};
```

contact_advanced.cpp#endif

```
#include "contact_advanced.h"

// Инициализация статических счётчиков
std::atomic<int> ContactAdvanced::news{0};
```

```
std::atomic<int> ContactAdvanced::constructed{0};

std::atomic<int> ContactAdvanced::copied{0};

std::atomic<int> ContactAdvanced::moved{0};

std::atomic<int> ContactAdvanced::destroyed{0};
```

main.cpp (демонстрация работы)

```
#include "contact_advanced.h"

#include <vector>


int main() {

    ContactAdvanced::resetCounters();


    std::vector<ContactAdvanced> v;

    v.reserve(3); // избегаем лишних копирований

    v.emplace_back();

    v.emplace_back();

    v.emplace_back();


    // Копия (тест copy constructor)

    ContactAdvanced copy = v[0];


    // Перемещение (тест move constructor)

    ContactAdvanced moved = std::move(v[1]);
```

```
    ContactAdvanced::printCounters();

    return 0;

}
```

Что делает этот код

- Каждый раз, когда создаётся объект `ContactAdvanced`, увеличивается счётчик `constructed`.

- При копировании (`ContactAdvanced(const ContactAdvanced&)`) — счётчик `copied`.

- При перемещении (`ContactAdvanced(ContactAdvanced&&)`) — счётчик `moved`.

- При удалении (`~ContactAdvanced()`) — `destroyed`.

- Переопределённый `operator new` считает количество выделений памяти (`news`).

Таким образом, можно:

1- Оценить, сколько раз реально создаются/копируются объекты.
2- Оптимизировать код, уменьшая количество копий (например, использовать `emplace_back` вместо `push_back`).

Пример вывода программы

```
Operator new calls: 3

Constructed: 3

Copied: 1

Moved: 1

Destroyed: 5
```

(цифры зависят от компилятора и оптимизаций)