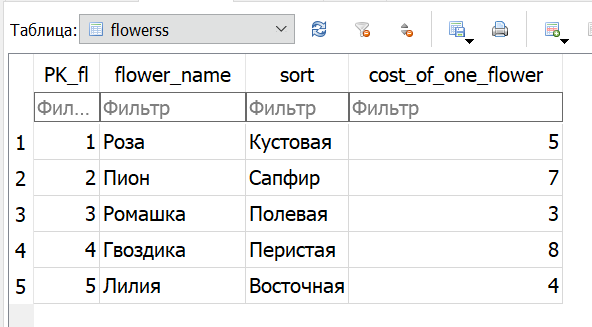
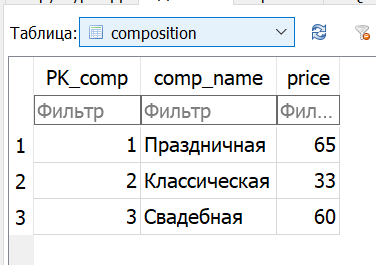
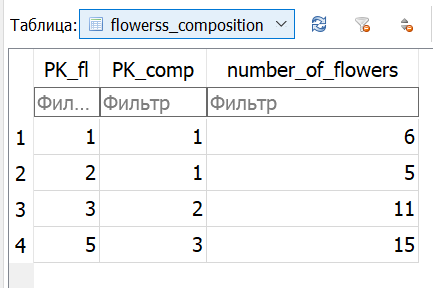
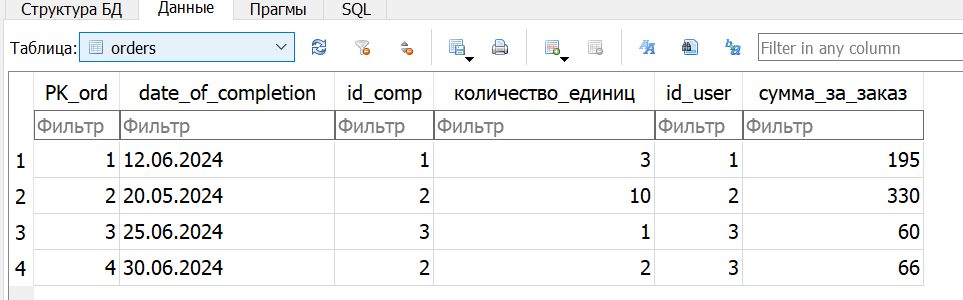
**ЛАБОРАТОРНАЯ РАБОТА 2**

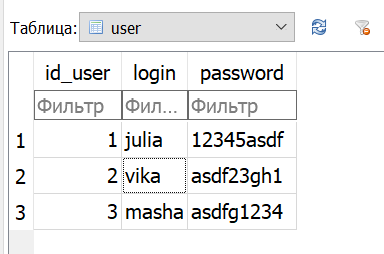
У нас есть 5 следующих таблиц:











Нам нужно написать оконное приложение на QT, которое позволяет вывести на экран каждую из этих таблиц по отдельности. Должна быть возможность обновить данные, удалить их и добавить. При вводе пароля в таблице ‘User’ должен быть валидатор.

#include <QApplication> // Предоставляет основной цикл событий приложения и обрабатывает взаимодействие с пользователем

#include <QSqlDatabase> // Предоставляет интерфейс для доступа и управления SQL-базами данных

#include <QSqlRelationalTableModel> // Предоставляет модель таблицы, которая поддерживает связи между таблицами

#include <QTableView> // Предоставляет представление для отображения данных из модели таблицы

#include <QVBoxLayout> // Предоставляет макет, который располагает дочерние виджеты вертикально

#include <QWidget> // Предоставляет базовый класс для всех пользовательских интерфейсных объектов

#include <QPushButton> // Предоставляет кнопку-команду

#include <QStackedWidget> // Предоставляет стек виджетов, где видим только один виджет в данный момент

#include <QDialog> // Предоставляет диалоговое окно, подходящее для модальных или немодальных диалогов

#include <QFormLayout> // Предоставляет макет, который располагает дочерние виджеты в виде формы

#include <QLineEdit> // Предоставляет однострочный текстовый редактор

#include <QRegularExpressionValidator> // Предоставляет валидатор, который проверяет, соответствует ли строка регулярному выражению

#include <QHBoxLayout> // Предоставляет макет, который располагает дочерние виджеты горизонтально

#include <QDebug> // Предоставляет удобный способ записи отладочных сообщений

// Функция проверки пароля на валидность

bool isValidPassword(const QString& password) {

if (password.length() != 9) { // Проверка длины пароля

return false;

}

int digitCount = 0;

bool hasNonDigit = false;

for (const QChar& c : password) { // Проверка на наличие цифр и нецифровых символов

if (c.isDigit()) {

digitCount++;

} else {

hasNonDigit = true;

}

}

return digitCount >= 3 && hasNonDigit; // Возвращает true, если пароль валиден

}

// Функция настройки базы данных

QSqlDatabase setupDatabase() {

QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");

db.setDatabaseName("D:\\data-base\\flowers.db"); // Указываем путь к базе данных

if (!db.open()) { // Проверка подключения к базе данных

qWarning() << "ERROR DB";

exit(1);

}

qDebug() << "Database connection established.";

return db;

}

// Функция создания кнопок для переключения таблиц

void createTableButtons(QVBoxLayout \*buttonLayout, QVector<QPushButton\*> &buttons, const QStringList &tableNames) {

for (const QString &tableName : tableNames) {

QPushButton\* button = new QPushButton(tableName); // Создание кнопки с названием таблицы

buttonLayout->addWidget(button); // Добавление кнопки в макет

buttons.append(button); // Добавление кнопки в вектор кнопок

}

}

// Функция создания вкладок с таблицами

void createTableTab(QStackedWidget \*stackedWidget, QSqlRelationalTableModel \*model, const QString &tableName, QSqlDatabase &db, QWidget &mainWindow) {

model->setTable(tableName); // Установка таблицы для модели

model->select(); // Выбор данных из таблицы

QTableView \*tableView = new QTableView;

tableView->setModel(model); // Установка модели для представления таблицы

tableView->resizeColumnsToContents(); // Подгонка размеров столбцов под содержимое

// Создание кнопки "Добавить"

QPushButton\* addButton = new QPushButton("Добавить");

QObject::connect(addButton, &QPushButton::clicked, [model, tableView, &mainWindow, tableName]() {

QDialog\* dialog = new QDialog(&mainWindow); // Создание диалогового окна для ввода данных

QFormLayout\* formLayout = new QFormLayout(dialog);

QVector<QLineEdit\*> lineEdits;

for (int j = 0; j < model->columnCount(); ++j) {

QLineEdit\* lineEdit = new QLineEdit;

formLayout->addRow(model->headerData(j, Qt::Horizontal).toString(), lineEdit);

lineEdits.append(lineEdit);

// Добавление валидатора для поля "password" в таблице "user"

if (tableName == "user" && model->headerData(j, Qt::Horizontal).toString().toLower() == "password") {

lineEdit->setValidator(new QRegularExpressionValidator(QRegularExpression(".{8,}")));

}

}

QPushButton\* saveButton = new QPushButton("Сохранить");

QObject::connect(saveButton, &QPushButton::clicked, [model, lineEdits, dialog, tableName]() {

// Проверка валидности пароля для таблицы "user"

if (tableName == "user") {

for (int j = 0; j < model->columnCount(); ++j) {

if (model->headerData(j, Qt::Horizontal).toString().toLower() == "password") {

if (!isValidPassword(lineEdits[j]->text())) {

qWarning() << "Пароль должен содержать минимум 9 символов и минимум 3 цифры.";

return;

}

}

}

}

int newRow = model->rowCount();

model->insertRow(newRow); // Вставка новой строки в модель

for (int j = 0; j < lineEdits.size(); ++j) {

model->setData(model->index(newRow, j), lineEdits[j]->text()); // Заполнение новой строки данными из полей ввода

}

model->submitAll(); // Сохранение изменений в модели

dialog->close(); // Закрытие диалогового окна

});

formLayout->addWidget(saveButton);

dialog->exec(); // Запуск диалогового окна

});

// Создание кнопки "Удалить"

QPushButton\* deleteButton = new QPushButton("Удалить");

QObject::connect(deleteButton, &QPushButton::clicked, [model, tableView]() {

QItemSelectionModel\* selectionModel = tableView->selectionModel(); // Получение модели выделения

QModelIndexList selectedRows = selectionModel->selectedRows(); // Получение списка выделенных строк

for (int i = selectedRows.count() - 1; i >= 0; --i) {

model->removeRow(selectedRows.at(i).row()); // Удаление выделенных строк из модели

}

model->select(); // Обновление данных в модели

});

// Создание кнопки "Обновить"

QPushButton\* updateButton = new QPushButton("Обновить");

QObject::connect(updateButton, &QPushButton::clicked, [model, tableView]() {

model->submitAll(); // Сохранение всех изменений в модели

model->select(); // Обновление данных в таблице

tableView->resizeColumnsToContents(); // Подгонка размеров столбцов под содержимое

});

// Создание макета для таблицы и кнопок

QVBoxLayout \*tableLayout = new QVBoxLayout;

tableLayout->addWidget(tableView);

tableLayout->addWidget(addButton);

tableLayout->addWidget(deleteButton);

tableLayout->addWidget(updateButton);

QWidget \*tabWidget = new QWidget;

tabWidget->setLayout(tableLayout); // Установка макета для виджета вкладки

stackedWidget->addWidget(tabWidget); // Добавление вкладки в стек виджетов

}

// Функция настройки главного окна

void setupMainWindow(QWidget &mainWindow, QVBoxLayout \*layout, QVector<QPushButton\*> &buttons, QStackedWidget \*stackedWidget, const QStringList &tableNames, QSqlDatabase &db) {

QVBoxLayout \*buttonLayout = new QVBoxLayout();

createTableButtons(buttonLayout, buttons, tableNames); // Создание кнопок для переключения таблиц

for (int i = 0; i < tableNames.size(); ++i) {

QSqlRelationalTableModel \*model = new QSqlRelationalTableModel(nullptr, db);

createTableTab(stackedWidget, model, tableNames[i], db, mainWindow); // Создание вкладок с таблицами

}

for (int i = 0; i < buttons.size(); ++i) {

QObject::connect(buttons[i], &QPushButton::clicked, [stackedWidget, i]() {

stackedWidget->setCurrentIndex(i); // Переключение вкладки при нажатии на кнопку

stackedWidget->show();

});

}

layout->addLayout(buttonLayout); // Добавление макета с кнопками в главный макет

layout->addWidget(stackedWidget); // Добавление стекового виджета в главный макет

stackedWidget->hide(); // Изначально скрываем стековый виджет

mainWindow.setLayout(layout); // Установка главного макета для главного окна

mainWindow.setWindowTitle("Database Viewer"); // Установка заголовка главного окна

mainWindow.resize(800, 600); // Установка размера главного окна

}

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

QApplication app(argc, argv);

QSqlDatabase db = setupDatabase(); // Настройка базы данных

QWidget mainWindow;

QVBoxLayout \*layout = new QVBoxLayout(&mainWindow);

QStackedWidget \*stackedWidget = new QStackedWidget;

QVector<QPushButton\*> buttons;

QStringList tableNames = {"flowerss", "composition", "flowerss\_composition", "user", "orders"};

setupMainWindow(mainWindow, layout, buttons, stackedWidget, tableNames, db); // Настройка главного окна

mainWindow.show(); // Показываем главное окно

return app.exec(); // Запуск основного цикла приложения

}

Юнит-тестирование – это процесс, который позволяет проверить работоспособность отдельных частей исходного кода программы.