Министерство связи и массовых коммуникаций Российской Федерации (Минкомсвязь РФ)  
Федеральное государственное образовательное бюджетное учреждение высшего профессионального образования  
"Сибирский государственный университет телекоммуникаций и информатики" (ФГОБУ ВПО "СибГУТИ")

*Кафедра ТС и ВС*

Лабораторная работа № 6

Создание базы данных в Qt

**Выполнили:** студенты группы *ИП-513*

*Майоров С.А.*

*Санин И.В.*

**Проверил:** старший преподаватель кафедры ТС и ВС

Лапова С.Г.

Новосибирск, 2017

**1. Цель работы**

Создание базы данных, концепция интервью в QtCreator.

**2. Листинг**

**mainwindow.h:**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "crypter.h"

#include "dialog.h"

#include "database.h"

#include "dbwindow.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private slots:

void on\_remember\_clicked();

void on\_pushButtonNext\_clicked();

void on\_linelogin\_editingFinished();

void on\_linepass\_editingFinished();

void updateMap();

private:

Ui::MainWindow \*ui;

DataBase \*db;

QMap< QPair<QString, QString>, QPair<QString, QString> > \*map;

};

#endif // MAINWINDOW\_H

**mainwindow.cpp:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QMessageBox>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

ui->labelWrong->hide();

db = new DataBase(this);

db->connectToDataBase("..\\..\\resources\\databases\\lab1.sqlite");

map = new QMap< QPair<QString, QString>, QPair<QString, QString> >();

this->updateMap();

}

MainWindow::~*MainWindow*()

{

delete ui;

}

void MainWindow::on\_remember\_clicked()

{

Dialog \*dial = new Dialog(map);

dial->show();

}

void MainWindow::on\_pushButtonNext\_clicked()

{

bool flag = false;

typedef QPair<QString, QString> item\_type;

foreach (item\_type item, map->keys()) {

if(ui->linelogin->text() == item.first)

{

if (ui->linepass->text() == item.second)

{

flag = true;

}

break;

}

}

if (!flag)

{

ui->labelWrong->show();

return;

}

DBWindow\* win = new DBWindow(ui->linelogin->text(), ui->linepass->text(), db->getDB(), this);

win->show();

this->hide();

ui->labelWrong->hide();

connect(win, SIGNAL(newUserAdded()), this, SLOT(updateMap()));

}

void MainWindow::on\_linelogin\_editingFinished()

{

if(ui->linepass->text() == "")

ui->linepass->setFocus();

}

void MainWindow::on\_linepass\_editingFinished()

{

if(ui->linelogin->text() == "")

ui->linelogin->setFocus();

}

void MainWindow::updateMap()

{

QSqlQuery\* query = new QSqlQuery(db->getDB());

query->exec("SELECT " TABLE2\_LOG ", " TABLE2\_PAS ", " TABLE2\_QST ", " TABLE2\_ANS " FROM " TABLE2);

if (!query->isActive())

QMessageBox::warning(

this,

tr("Database Error"),

query->lastError().text()

);

while (query->next())

{

QString log = Crypter::decryptString(query->value(0).toString());

QString pas = Crypter::decryptString(query->value(1).toString());

QPair<QString, QString>\* logNpass = new QPair<QString, QString>(log, pas);

QString qst = Crypter::decryptString(query->value(2).toString());

QString ans = Crypter::decryptString(query->value(3).toString());

QPair<QString, QString>\* questNans = new QPair<QString, QString>(qst, ans);

map->insert(\*logNpass, \*questNans);

qDebug() << log << " " << pas << " " << qst << " " << ans << endl;

}

}

**dbwindow.cpp:**

#include "dbwindow.h"

#include "ui\_dbwindow.h"

DBWindow::DBWindow(QString login, QString password, QSqlDatabase db, QWidget \*parent) :

QMainWindow(parent),

log(login),

pass(password),

db(db),

ui(new Ui::DBWindow)

{

ui->setupUi(this);

if (login == "admin") ui->tableView->setEditTriggers(QAbstractItemView::DoubleClicked);

else

{

ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);

ui->actionNewUser->setVisible(false);

}

ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);

ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

mod = new QSqlTableModel();

mod->*setTable*(TABLE1);

mod->*setHeaderData*(1, Qt::Horizontal, QObject::tr("ФИО"));

ui->comboBox->addItem("ФИО");

mod->*setHeaderData*(2, Qt::Horizontal, QObject::tr("Пол"));

ui->comboBox->addItem("Пол");

mod->*setHeaderData*(3, Qt::Horizontal, QObject::tr("Мобильный телефон"));

ui->comboBox->addItem("Мобильный телефон");

mod->*setHeaderData*(4, Qt::Horizontal, QObject::tr("Дата регистрации абонемента"));

ui->comboBox->addItem("Дата регистрации абонемента");

mod->*setHeaderData*(5, Qt::Horizontal, QObject::tr("Длительность абонемента"));

ui->comboBox->addItem("Длительность абонемента");

mod->*setHeaderData*(6, Qt::Horizontal, QObject::tr("Прошлые абонементы"));

ui->comboBox->addItem("Прошлые абонементы");

mod->*setHeaderData*(7, Qt::Horizontal, QObject::tr("Справка"), Qt::DecorationRole);

mod->*setHeaderData*(8, Qt::Horizontal, QObject::tr("Длительность справки"));

mod->*select*();

ui->tableView->*setModel*(mod);

ui->tableView->setColumnHidden(0, true);

ui->tableView->setAlternatingRowColors(true);

ui->tableView->resizeColumnsToContents();

ui->tableView->resizeRowsToContents();

}

DBWindow::~*DBWindow*()

{

delete ui;

}

void DBWindow::on\_actionExit\_triggered()

{

((QWidget \*) this->parent())->show();

this->close();

}

void DBWindow::on\_lineEdit\_CharSearch\_textChanged(const QString &arg1)

{

int index = ui->comboBox->currentIndex() + 1;

ui->tableView->setSelectionMode(QAbstractItemView::MultiSelection);

ui->tableView->clearSelection();

for (int i = 0; i < mod->*rowCount*(); ++i){

ui->tableView->showRow(i);

QString str = mod->*index*(i,index).data().toString();

if (!str.startsWith(arg1, Qt::CaseInsensitive)) ui->tableView->hideRow(i);

else ui->tableView->showRow(i);

}

if (arg1 == "")

ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);

}

void DBWindow::on\_actionNewUser\_triggered()

{

bool ok;

QString login = QInputDialog::getText(this, "Новый пользователь (логин)", "Введите логин", QLineEdit::Normal, "", &ok);

if (!ok || login.isEmpty())

return;

QString password = QInputDialog::getText(this, "Новый пользователь (пароль)", "Введите пароль", QLineEdit::Password, "", &ok);

if (!ok || password.isEmpty())

return;

QString question = QInputDialog::getText(this, "Новый пользователь (вопрос)", "Введите контрольный вопрос", QLineEdit::Normal, "", &ok);

if (!ok || question.isEmpty())

return;

QString answer = QInputDialog::getText(this, "Новый пользователь (ответ)", "Введите ответ на контрольный вопрос", QLineEdit::Normal, "", &ok);

if (!ok || answer.isEmpty())

return;

QSqlQuery\* query = new QSqlQuery(db);

query->prepare(QString("INSERT INTO " TABLE2 " VALUES(null, '%1', '%2', '%3', '%4');")

.arg(Crypter::cryptString(login))

.arg(Crypter::cryptString(password))

.arg(Crypter::cryptString(question))

.arg(Crypter::cryptString(answer))

);

qDebug() << query->lastQuery() << endl;

query->exec();

if (!query->isActive())

QMessageBox::warning(

this,

tr("Database Error"),

query->lastError().text()

);

emit newUserAdded();

while (query->next())

{

qDebug() << "hm..." << endl;

}

}

void DBWindow::on\_actionChangePassword\_triggered()

{

bool ok;

QString oldPassword = QInputDialog::getText(this, "Новый пароль", "Введите старый пароль", QLineEdit::Password, "", &ok);

if (!ok)

return;

if (oldPassword != pass)

{

QMessageBox::warning(

this,

tr("Неверный пароль"),

tr("Неверный пароль")

);

return;

}

while (true)

{

QString password1 = QInputDialog::getText(this, "Новый пароль", "Введите новый пароль", QLineEdit::Password, "", &ok);

if (!ok) return;

if (password1.isEmpty())

{

QMessageBox::warning(

this,

tr("Неверный пароль"),

tr("Пароль не может быть пустым")

);

continue;

}

QString password2 = QInputDialog::getText(this, "Новый пароль", "Еще раз введите новый пароль", QLineEdit::Password, "", &ok);

if (!ok) return;

if (password1 != password2)

{

QMessageBox::warning(

this,

tr("Неверный пароль"),

tr("Введенные пароли не совпадают")

);

continue;

}

pass = password1;

break;

}

QSqlQuery\* query = new QSqlQuery(db);

query->exec("UPDATE " TABLE2 " SET " TABLE2\_PAS " = '" + Crypter::cryptString(pass) + "' WHERE " TABLE2\_LOG " = '" + Crypter::cryptString(log) + "';");

if (!query->isActive())

QMessageBox::warning(

this,

tr("Database Error"),

query->lastError().text()

);

while (query->next())

{

qDebug() << "hm..." << endl;

}

emit newUserAdded();

}

**dialog.cpp:**

#include "dialog.h"

#include "ui\_dialog.h"

Dialog::Dialog(QMap<QPair<QString, QString>, QPair<QString, QString> > \*map, QWidget \*parent) :

QDialog(parent),

ui(new Ui::Dialog),

map(map)

{

ui->setupUi(this);

ui->pass->hide();

ui->question->hide();

ui->linequestion->hide();

ui->finish->hide();

}

Dialog::~*Dialog*()

{

delete ui;

}

void Dialog::on\_linelogin\_editingFinished()

{

bool flag = false;

typedef QPair<QString, QString> item\_type;

foreach (item\_type item, map->keys()) {

if(ui->linelogin->text() == item.first)

{

flag = true;

login = item.first;

password = item.second;

question = map->find(item).value().first;

answer = map->find(item).value().second;

break;

}

}

if (flag)

{

ui->question->setText(question);

ui->linequestion->show();

ui->pass->hide();

ui->finish->hide();

} else {

ui->question->setText("Incorrect login");

ui->linequestion->hide();

ui->pass->hide();

ui->finish->hide();

}

ui->question->show();

}

void Dialog::on\_linequestion\_editingFinished()

{

bool flag = false;

if (ui->linequestion->text() == answer)

flag = true;

if (flag)

{

ui->pass->setText(password);

ui->finish->show();

} else {

ui->pass->setText("Incorrect answer");

ui->finish->hide();

}

ui->pass->show();

}

void Dialog::on\_finish\_clicked()

{

this->*accept*();

}

**3. Результат работы**





