МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

(БГТУ им. В.Г.Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Базы данных

Лабораторная работа №6

Приложение для взаимодействия с СУБД

Выполнил:

ст. гр. ВТ-31

Шевченко К.К.

Проверил:

ст.пр. Панченко М.В.

Белгород, 2019

**Выполнение**

Код программы

Файл “main.cpp”

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}

Файл “insertform.h”

#ifndef INSERTFORM\_H

#define INSERTFORM\_H

#include <QDialog>

#include <QStringList>

#include <QLabel>

#include <QLineEdit>

#include <QMessageBox>

#include <QVector>

namespace **Ui** {

class **InsertForm**;

}

class **InsertForm** : public QDialog

{

Q\_OBJECT

public:

explicit **InsertForm**(QWidget \*parent = nullptr);

~***InsertForm***();

void **setColumns**(const QStringList &list);

QStringList **getresultList**();

private slots:

void **on\_pushButton\_clicked**();

void **on\_pushButton\_2\_clicked**();

private:

Ui::InsertForm \*ui;

QVector<QLineEdit\*> line;

QStringList res\_list;

};

#endif // INSERTFORM\_H

Файл “insertform.cpp”

#include "insertform.h"

#include "ui\_insertform.h"

InsertForm::**InsertForm**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::InsertForm)

{

ui->setupUi(this);

}

InsertForm::~***InsertForm***(){

delete ui;

}

void InsertForm::**setColumns**(const QStringList &list){

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

line << new QLineEdit;

ui->formLayout->addRow(new QLabel(list[i]), line[i]);

}

}

QStringList InsertForm::**getresultList**(){

return res\_list;

}

void InsertForm::**on\_pushButton\_clicked**(){

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

if (line[i]->text().size() == 0){

res\_list.clear();

QMessageBox::warning(this, "", "Заполните все поля");

return;

}

res\_list << line[i]->text();

}

*accept*();

}

void InsertForm::**on\_pushButton\_2\_clicked**(){

*reject*();

}

Файл “exportsql.h”

#ifndef EXPORTSQL\_H

#define EXPORTSQL\_H

#include <QJsonObject>

#include <QJsonArray>

#include <QJsonDocument>

#include <QStringList>

#include <QVector>

#include <QFile>

class **ExportSQLtoJSON**{

QString t\_name;

QStringList headers;

QVector<QStringList> rows;

public:

**ExportSQLtoJSON**(){}

void **setData**(const QString &t\_name, const QStringList &headers, const QVector<QStringList> &rows);

bool **exportToFile**(const QString &f\_name);

};

class **ExportSQLtoCSV**{

QString t\_name;

QStringList headers;

QVector<QStringList> rows;

public:

**ExportSQLtoCSV**(){}

void **setData**(const QStringList &headers, const QVector<QStringList> &rows);

void **setData**(const QVector<QStringList> &rows);

bool **exportToFile**(const QString &f\_name, char separator = ',', bool with\_headers = false);

};

#endif // EXPORTSQL\_H

Файл “exportsql.cpp”

#include "exportsql.h"

#include <QDebug>

void ExportSQLtoJSON::**setData**(const QString &t\_name, const QStringList &headers, const QVector<QStringList> &rows){

this->t\_name = t\_name;

this->headers = headers;

this->rows = rows;

}

bool ExportSQLtoJSON::**exportToFile**(const QString &f\_name){

QJsonObject j\_table\_row;

QJsonArray j\_table\_arr;

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

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

j\_table\_row[headers[j]] = rows[i][j];

j\_table\_arr.append(j\_table\_row);

}

QJsonObject j\_result;

j\_result[t\_name] = j\_table\_arr;

QFile f(f\_name);

if (!f.*open*(QIODevice::WriteOnly)) return false;

if (f.write(QJsonDocument(j\_result).toJson(QJsonDocument::Indented)) == 0){

f.remove();

return false;

}

f.*close*();

return true;

}

void ExportSQLtoCSV::**setData**(const QStringList &headers, const QVector<QStringList> &rows){

this->headers = headers;

this->rows = rows;

}

void ExportSQLtoCSV::**setData**(const QVector<QStringList> &rows){

this->rows = rows;

}

bool ExportSQLtoCSV::**exportToFile**(const QString &f\_name, char separator, bool with\_headers){

QString res\_str;

if (with\_headers){

for (int i = 0; i < headers.size() - 1; i++)

res\_str += headers[i] + separator;

res\_str += headers[headers.size() - 1] + "\n";

}

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

for (int j = 0; j < rows[i].size() - 1; j++)

res\_str += rows[i][j] + separator;

res\_str += rows[i][rows[i].size() - 1] + "\n";

}

QFile f(f\_name);

if (!f.*open*(QIODevice::WriteOnly)) return false;

if (!f.write(res\_str.toUtf8())){

f.remove();

return false;

}

f.*close*();

return true;

}

Файл “mainwindow.h”

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QSqlDatabase>

#include <QMessageBox>

#include <QSqlError>

#include <QSqlQuery>

#include <QTableWidget>

#include "insertform.h"

#include <QStringList>

#include "exportsql.h"

#define IS\_DOUBLE\_CLICK 0

#define IS\_BUTTON 1

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

bool **connect**();

void **disconnect**();

void **showTables**();

void **showTable**(QString t\_name);

void **insertRow**(QString t\_name);

void **removeRow**(QString t\_name, int row);

QString **getCondition**(QString t\_name, int row);

void **changeField**(QString t\_name, int row, int col, int method);

void **prepareToExport**(QTableWidget \*table, QStringList \*headers, QVector<QStringList> \*rows);

private slots:

void **on\_main\_table\_cellClicked**(int row, int column);

void **on\_button\_to\_table\_clicked**();

void **on\_button\_back\_clicked**();

void **on\_insert\_button\_clicked**();

void **on\_remove\_button\_clicked**();

void **on\_change\_button\_clicked**();

void **on\_cur\_table\_cellClicked**(int row, int column);

void **on\_cur\_table\_cellDoubleClicked**(int row, int column);

void **on\_cur\_table\_cellChanged**(int row, int column);

void **on\_button\_exp\_to\_json\_clicked**();

void **on\_button\_exp\_to\_csv\_clicked**();

private:

Ui::MainWindow \*ui;

QSqlDatabase db;

int cl\_row, cl\_col;

int cur\_cl\_row, cur\_cl\_col;

QString cur\_table\_name;

bool change\_flag;

};

#endif // MAINWINDOW\_H

Файл “mainwindow.cpp”

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QDebug>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

bool b = connect();

if (b) showTables();

cl\_row = cl\_col = -1;

cur\_cl\_row = cur\_cl\_col = -1;

ui->stackedWidget->setCurrentWidget(ui->main\_page);

}

MainWindow::~***MainWindow***()

{

disconnect();

delete ui;

}

bool MainWindow::**connect**(){

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

db.setHostName("localhost");

//db.setHostName("192.168.0.195");

//db.setPort(5432);

db.setDatabaseName("postgres");

db.setUserName("postgres");

db.setPassword("Kostya1234");

bool ok = db.open();

if (!ok) {qDebug() << "try";QMessageBox::critical(this, "Ошибка подключения к БД!", db.lastError().databaseText() + "\n" + db.lastError().driverText());}

else ui->statusBar->showMessage("Подключение выполнено успешно", 3000);

return ok;

}

void MainWindow::**disconnect**(){

db.close();

}

void MainWindow::**showTables**(){

QSqlQuery q;

q.exec("select table\_name from information\_schema.tables where table\_schema = \'public\';");

QTableWidget \*t = ui->main\_table;

QTableWidgetItem \*item;

item = new QTableWidgetItem("Имя таблицы");

t->setColumnCount(1);

t->setHorizontalHeaderItem(0, item);

t->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

int i = 0;

while (q.next()){

item = new QTableWidgetItem(q.value(0).toString());

t->insertRow(i);

t->setItem(i++, 0, item);

}

}

void MainWindow::**showTable**(QString t\_name){

ui->stackedWidget->setCurrentWidget(ui->page\_2);

QTableWidget \*t = ui->cur\_table;

t->clear();

t->setRowCount(0);

t->setColumnCount(0);

QString request = "select column\_name from information\_schema.columns where table\_name = \'" + t\_name + "\' order by ordinal\_position";

QSqlQuery q;

if (!q.exec(request))

ui->statusBar->showMessage(q.lastError().databaseText() + "; " + q.lastError().driverText());

while(q.next()){

t->insertColumn(t->columnCount());

t->setHorizontalHeaderItem(t->columnCount() - 1, new QTableWidgetItem(q.value(0).toString()));

}

t->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

request = "select \* from " + t\_name;

if (!q.exec(request))

ui->statusBar->showMessage(q.lastError().databaseText() + "; " + q.lastError().driverText());

while(q.next()){

t->insertRow(t->rowCount());

for (int i = 0; i < t->columnCount(); i++){

t->setItem(t->rowCount()-1, i, new QTableWidgetItem(q.value(i).toString()));

}

}

}

void MainWindow::**insertRow**(QString t\_name){

QString request = "select column\_name, is\_identity from information\_schema.columns where table\_name = \'" + t\_name + "\' order by ordinal\_position";

QSqlQuery q;

q.exec(request);

QStringList list;

while (q.next())

if (q.value(1) != "YES")

list << q.value(0).toString();

InsertForm form;

form.setColumns(list);

if (!form.*exec*()) return;

list.clear();

list = form.getresultList();

QString command = "insert into " + t\_name + " (";

q.first();

int i = q.size()-1;

while(i--){

if (q.value(1) != "YES")

command += q.value(0).toString() + ", ";

q.next();

}

command += q.value(0).toString() + ") values (\'" + list[0] + "\'";

for (int i = 1; i < list.size(); i++)

command += ", \'" + list[i] + "\'";

command += ");";

if (q.exec(command)){

ui->statusBar->showMessage("Добавление записи выполнено успешно", 3000);

showTable(t\_name);

}

else ui->statusBar->showMessage(q.lastError().databaseText() + "; " + q.lastError().driverText());

}

QString MainWindow::**getCondition**(QString t\_name, int row){

QString request = "select column\_name from information\_schema.key\_column\_usage where table\_name = \'" + t\_name + "\' and constraint\_name like \'%\_pkey\';";

QString condition;

QSqlQuery q;

if(q.exec(request)){

q.first();

QTableWidget \*t = ui->cur\_table;

int pkey\_col\_num = -1;

for (int i = 0; i < t->columnCount(); i++)

if (t->horizontalHeaderItem(i)->text() == q.value(0).toString())

pkey\_col\_num = i;

condition = q.value(0).toString() + " = " + t->item(row, pkey\_col\_num)->text();

}

return condition;

}

void MainWindow::**removeRow**(QString t\_name, int row){

QString condition = getCondition(t\_name, row);

QString request = "delete from " + t\_name + " where " + condition + ";";

QSqlQuery q;

if (q.exec(request)){

ui->statusBar->showMessage("Удаление записи выполнено успешно", 3000);

showTable(t\_name);

}

else ui->statusBar->showMessage(q.lastError().databaseText() + "; " + q.lastError().driverText());

}

void MainWindow::**changeField**(QString t\_name, int row, int col, int method){

change\_flag = false;

QString condition = getCondition(t\_name, row);

QTableWidget \*t = ui->cur\_table;

QString data;

if (method == IS\_DOUBLE\_CLICK) data = t->item(row, col)->text();

if (method == IS\_BUTTON){

QStringList list;

list << t->horizontalHeaderItem(col)->text();

InsertForm form;

form.setColumns(list);

if (!form.*exec*()) return;

list.clear();

list = form.getresultList();

data = list[0];

}

QString request = "update " + t\_name + " set " + t->horizontalHeaderItem(col)->text() + " = \'" + data + "\' where " + condition + ";";

QSqlQuery q;

if (q.exec(request)){

ui->statusBar->showMessage("Изменение поля выполнено успешно", 3000);

showTable(t\_name);

}

else ui->statusBar->showMessage(q.lastError().databaseText() + "; " + q.lastError().driverText());

showTable(t\_name);

}

void MainWindow::**prepareToExport**(QTableWidget \*table, QStringList \*headers, QVector<QStringList> \*rows){

if (headers != nullptr)

for (int i = 0; i < table->columnCount(); i++)

\*headers << table->horizontalHeaderItem(i)->text();

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

QStringList list;

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

list << table->item(i, j)->text();

\*rows << list;

}

}

//---------------------------------------------------------------------------------------------------------------

void MainWindow::**on\_main\_table\_cellClicked**(int row, int column){

cl\_row = row;

cl\_col = column;

}

void MainWindow::**on\_cur\_table\_cellClicked**(int row, int column){

cur\_cl\_row = row;

cur\_cl\_col = column;

}

void MainWindow::**on\_button\_to\_table\_clicked**(){

if (cl\_row == -1 || cl\_col == -1) return;

cur\_table\_name = ui->main\_table->item(cl\_row, cl\_col)->text();

change\_flag = false;

cl\_row = cl\_col = -1;

cur\_cl\_col = cur\_cl\_row = -1;

showTable(cur\_table\_name);

ui->label\_cut\_table\_name->setText(cur\_table\_name);

}

void MainWindow::**on\_button\_back\_clicked**(){

ui->stackedWidget->setCurrentWidget(ui->main\_page);

cur\_table\_name.clear();

cur\_cl\_col = cur\_cl\_row = -1;

}

void MainWindow::**on\_insert\_button\_clicked**(){

insertRow(cur\_table\_name);

}

void MainWindow::**on\_remove\_button\_clicked**(){

if (cur\_cl\_col == -1 || cur\_cl\_row == -1) return;

removeRow(cur\_table\_name, cur\_cl\_row);

cur\_cl\_row = cur\_cl\_col = -1;

}

void MainWindow::**on\_change\_button\_clicked**(){

if (cur\_cl\_col == -1 || cur\_cl\_row == -1) return;

change\_flag = false;

changeField(cur\_table\_name, cur\_cl\_row, cur\_cl\_col, IS\_BUTTON);

cur\_cl\_row = cur\_cl\_col = -1;

}

void MainWindow::**on\_cur\_table\_cellDoubleClicked**(int row, int column){

change\_flag = true;

cur\_cl\_row = cur\_cl\_col = -1;

}

void MainWindow::**on\_cur\_table\_cellChanged**(int row, int column){

if (!change\_flag) return;

changeField(cur\_table\_name, row, column, IS\_DOUBLE\_CLICK);

change\_flag = false;

cur\_cl\_row = cur\_cl\_col = -1;

}

void MainWindow::**on\_button\_exp\_to\_json\_clicked**(){

QStringList headers;

QVector<QStringList> rows;

prepareToExport(ui->cur\_table, &headers, &rows);

ExportSQLtoJSON exp;

exp.setData(cur\_table\_name, headers, rows);

if (exp.exportToFile(cur\_table\_name + ".json"))

ui->statusBar->showMessage("Экспорт в \*.json выполнен успешно, файл сохранён под именем " + cur\_table\_name + ".json");

else ui->statusBar->showMessage("Возникли ошибки при экспорте", 3000);

}

void MainWindow::**on\_button\_exp\_to\_csv\_clicked**(){

QVector<QStringList> rows;

prepareToExport(ui->cur\_table, nullptr, &rows);

ExportSQLtoCSV exp;

exp.setData(rows);

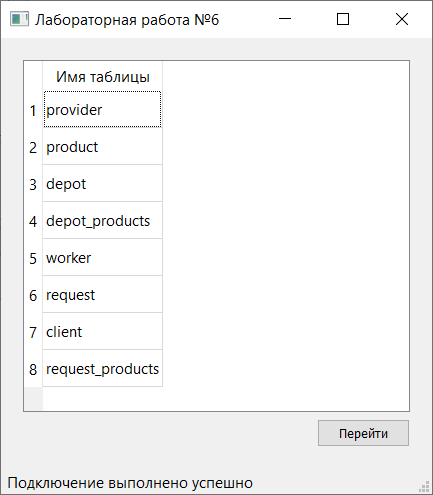
if (exp.exportToFile(cur\_table\_name + ".csv", ';'))

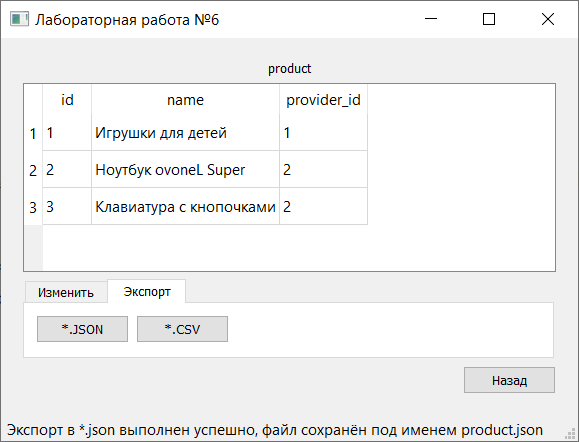
ui->statusBar->showMessage("Экспорт в \*.csv выполнен успешно, файл сохранён под именем " + cur\_table\_name + ".csv");

else ui->statusBar->showMessage("Возникли ошибки при экспорте", 3000);

}

**Результаты работы программы**





Содержимое файла product.json

