|  |  |
| --- | --- |
| Gerb-BMSTU_01**­­­­­­­­­** | **Министерство образования и науки Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ: Информатики и систем управления

КАФЕДРА: Компьютерные системы и сети

**ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ**

Студент Боярских Никита Игоревич

*фамилия, имя, отчество*

Группа ИУ6-22

Тип практики Практикум по программированию

Название предприятия МГТУ им. Н.Э. Баумана

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_Боярских Н.И.\_\_\_**

*подпись, дата фамилия, и.о.*

Руководитель практики **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_Черноусова Т.Г.\_\_\_**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2017 г.*

**Оглавление**

Введение 3

1. Основная часть 4

1.1 Главная форма4

1.2 Форма «Отчёт по стране»5

1.3 Форма «Отчёт импорта по товару» 5

1.4 Форма «График экспорта товара»6

2. Скриншоты программы4

Заключение 6

Список литературы 7

Приложения8

1) Главный модуль main.cpp8

2) Файл ресурсов practice\_3.rc8

3) Заголовочный файл модуля mainwindow.h9

4) Исходный код модуля mainwindow.cpp10

5) Заголовочный файл модуля validator\_delegate.h17

6) Исходный код модуля validator\_delegate.cpp18

8) Диаграмма классов программы 26

9) Диаграмма класса валидатора27

**Введение**

**Задание:**

Файл данных содержит следующие сведения: наименование товара, объем сделки, год заключения сделки, страна, направление (экспорт или импорт). Программа должна обеспечить возможность табличного ввода и в интерактивном режиме воспринимать каждый из перечисленных запросов и давать на него ответ.

* Определить все товары, импортируемые из данной страны и суммарный объем сделок (в у.е., которые пользователь выбирает в соответствии с типом упаковки и отгрузки товара (поштучный, упаковками, сыпучий, жидкий ит.д.))
* Определить страну, экспорт в которую имеет наибольший объем.
* Определить все страны, из которых импортируется данный товар, и объем сделок в каждом случае.
* Построить график экспорта заданного товара по годам.

**Цель работы** - выполнение объектной декомпозиции, разработка интерфейса, диаграмм состояний интерфейса, диаграмм классов интерфейсной и предметной областей: разработка, тестирование и отладка программы в среде QTCreator.

**1. Основная часть**

(Скриншоты интерфейса программы – см. приложение №7)

(Классы программы – см. приложение №8)

**Untitled Diagram.png**

Рис. 1 –объектная декомпозиция [2]

**1.1 Главная форма**

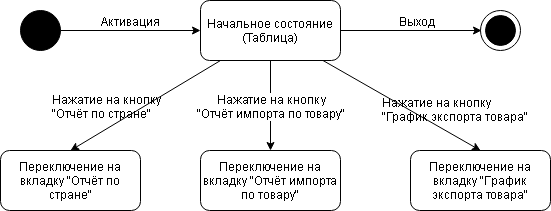


Рис. 2 – диаграмма состояния интерфейса программы [1,3]

1.2 Форма «Отчёт по стране»

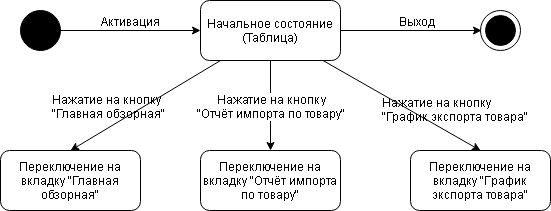


Рис. 3– диаграмма состояния интерфейса формы [1,3]

1.3 Форма «Отчёт импорта по товару»

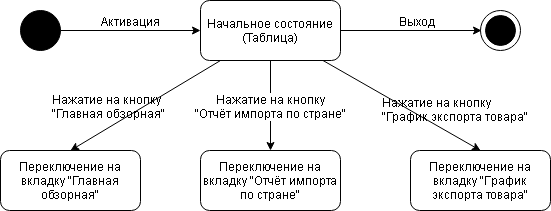


Рис. 4 – диаграмма состояния интерфейса формы [1,3]

1.4 Форма «График экспорта товара»

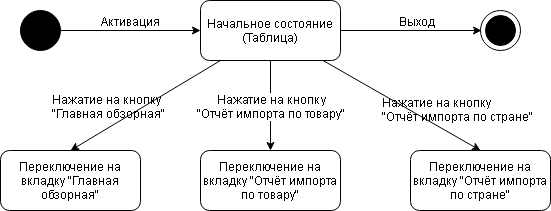


Рис. 5 – диаграмма состояния интерфейса формы [1,3]

**2. Скриншоты интерфейса программы**

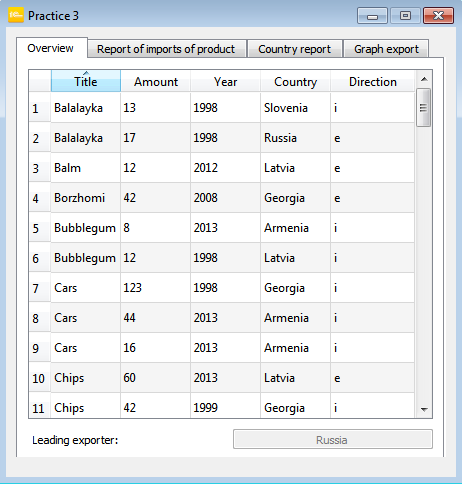


Рис. 6 – Вид главной формы (сортировка по названию – по умолчанию)

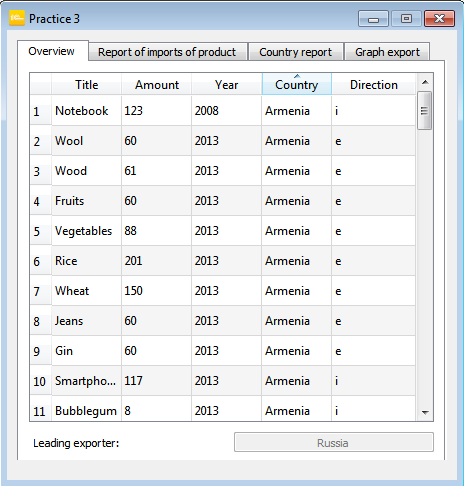


Рис. 7 – Вид главной формы (сортировка по стране)

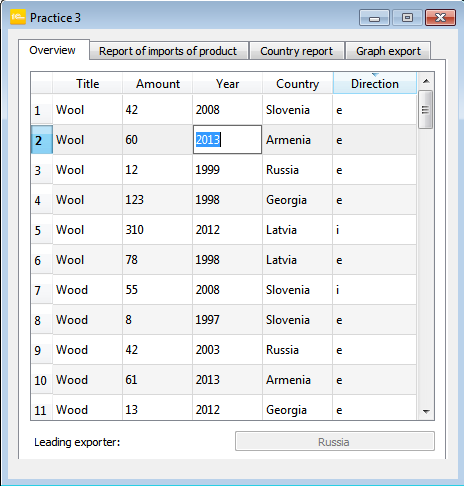


Рис. 8 – Вид главной формы (редакторование строки таблицы – поле «год»)

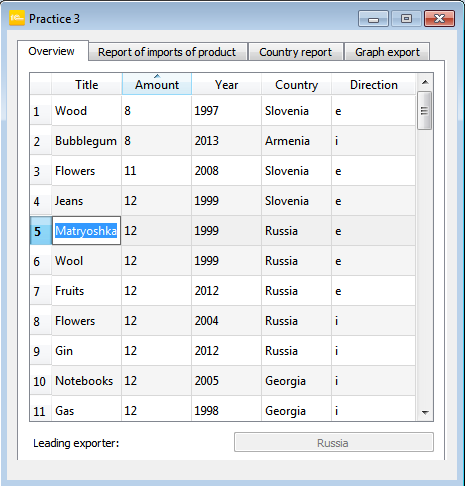


Рис. 9 – Вид главной формы (редакторование строки таблицы – поле «название»)

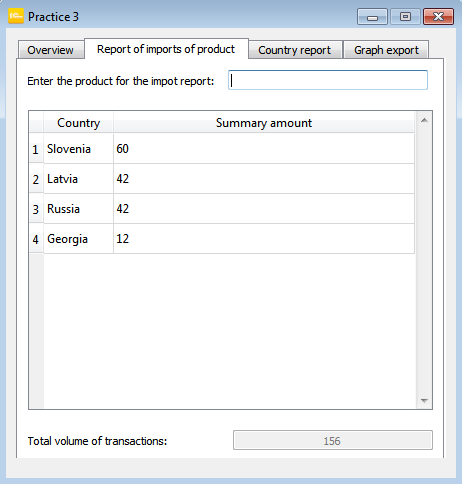


Рис. 10 – Вид формы отчёта по товару (товар – «Notebooks»)

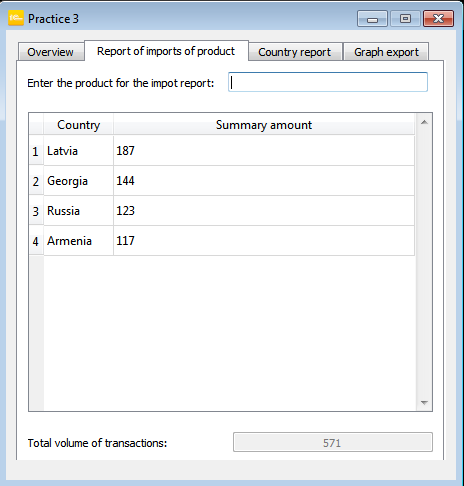


Рис. 11 – Вид формы отчёта по товару (товар – «Smartphones»)

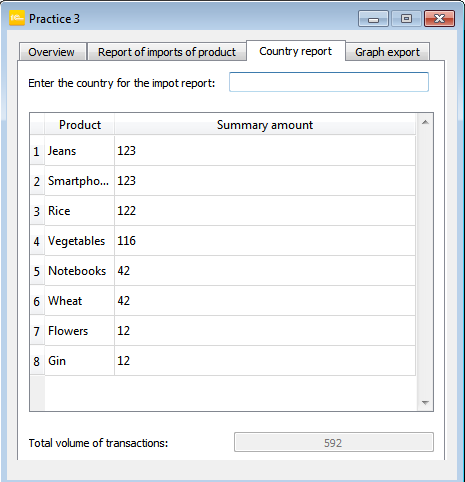


Рис. 12– Вид формы отчёта по стране (страна – «Russia»)

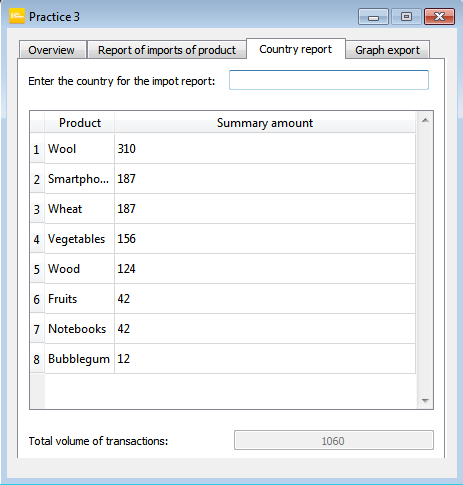


Рис. 13– Вид формы отчёта по стране (страна – «Latvia»)

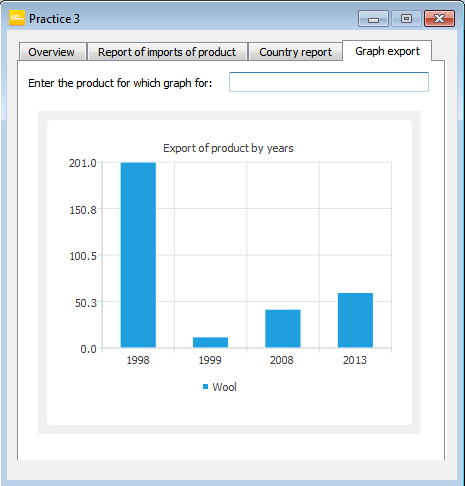


Рис. 14– Вид формы графика экспорта товара по годам (товар – «Wool»)

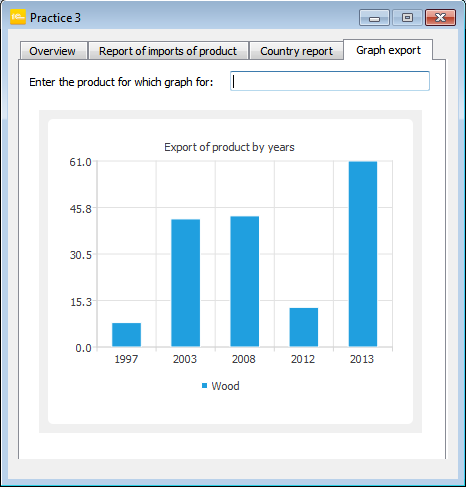


Рис. 15– Вид формы графика экспорта товара по годам (товар – «Wood»)

**Заключение**

Разработана объектная декомпозиция [1,2], диаграмма состояний интерфейса [1,3], диаграммы классов интерфейсной и предметной областей [1], диаграмма последовательностей каждой из реализуемых операций [1]. Создана и протестирована программа [1, 2, 3]. Тестирование показала корректность работы программы.

**Список литературы**

**Основная литература:**

1. Иванова Г.С. . C++. Часть 3. Создание графических интерфейсов пользователя с использованием библиотеки Qt 4.7: Учебное пособие. – М.: МГТУ им. Н.Э. Баумана, 2014. – 52 с.

**Дополнительная литература:**

1. Подбельский В.В. Стандартный Си++: Учеб.пособие. – М.: Финансы и статистика, 2008.
2. Иванова Г.С., Ничушкина Т.Н. Объектно-ориентированное программирование. Учеб.для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2014.
3. <http://www.cyberforum.ru/> (версия от 15.05.17)
4. <https://www.stackoverflow.com/> (версия от 15.05.17)

**Приложения**

1. **Главный модуль main.cpp [1]**

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

1. **Файл ресурсов practice\_3.rc [4]**

IDI\_ICON1 ICON "favicon.ico"

1. **Заголовочный файл модуля mainwindow.h [4, 5]**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QStandardItemModel>

#include <QItemSelection>

#include <QTableView>

#include <QtSql>

#include <QtCharts>

#include "validator\_delegate.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow( QWidget \*parent = 0 );

~MainWindow();

public slots:

void on\_tabBarMain\_currentChanged( int index );

void on\_iptImportProductReport\_returnPressed();

void on\_iptImportCountryReport\_returnPressed();

void on\_iptGraphExport\_returnPressed();

private:

Ui::MainWindow\* ui;

ValidatorDelegate\* delegate;

QSqlQueryModel\* model;

QSqlTableModel\* main\_model;

QSqlDatabase db;

QLabel\* lblGraphExport;

QLineEdit\* iptGraphExport;

QChartView\* chart\_view;

QBarCategoryAxis\* chart\_axis;

QChart\* chart;

QBarSeries\* chart\_series;

QBarSet\* chart\_set;

bool create\_main\_overview\_model();

QSqlError create\_import\_product\_report\_model();

QSqlError create\_import\_country\_report\_model();

bool create\_chart( QString title );

void update\_main\_overview();

void update\_import\_product\_report();

void update\_import\_country\_report();

void update\_graph\_export();

void setup\_table\_view( QTableView \*table\_view );

bool setup\_database( QString dbname );

};

#endif // MAINWINDOW\_H

1. **Исходный код модуля mainwindow.cpp [4, 5]**

#include <QMessageBox>

#include <QStandardItemModel>

#include <QItemSelection>

#include <QtSql>

#include <QtCharts>

#include <QTableView>

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "validator\_delegate.h"

MainWindow::MainWindow( QWidget \*parent ) :

QMainWindow( parent ),

ui( new Ui::MainWindow ),

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

{

// Setup UI

ui->setupUi(this);

ui->wdgGraphExport->setLayout(

new QBoxLayout( QBoxLayout::TopToBottom, ui->tabGraphExport )

);

// Setup dinamic variables

model = nullptr;

chart = nullptr;

chart\_series = nullptr;

chart\_axis = nullptr;

chart\_view = nullptr;

chart\_set = nullptr;

// Setup validator

delegate = new ValidatorDelegate;

ui->tblMainOverview->setItemDelegate( delegate );

// Sqlite file

QString filename = QString("../PR3/Practice3.sqlite");

// Setup DB

if( !setup\_database( filename ) ) {

QMessageBox::critical( this, tr("Error"), tr("Error: failed to setup database"));

exit(1);

}

// Setup Main Overview

update\_main\_overview();

}

(Продолжение кода на следующей странице)

(Продолжение кода)

MainWindow::~MainWindow()

{

db.close();

delete ui;

delete delegate;

delete model;

delete main\_model;

delete chart;

}

/// Slots

void MainWindow::on\_tabBarMain\_currentChanged( int index ) {

switch( index ) {

case 0: // Main Overview

update\_main\_overview();

break;

case 1: // Import product report

update\_import\_product\_report();

break;

case 2: // Import country report

update\_import\_country\_report();

break;

case 3: // Graph export

update\_graph\_export();

break;

}

// Cleanup

ui->statusBar->clearMessage();

}

void MainWindow::on\_iptImportProductReport\_returnPressed() {

QSqlError error = create\_import\_product\_report\_model();

if( error.isValid() ) {

QMessageBox::critical( this, tr("Error"), error.text() );

exit(1);

}

setup\_table\_view( ui->tblImportProductReport );

QString title = ui->iptImportProductReport->text();

update\_import\_product\_report();

ui->tblImportProductReport->setEnabled( true );

ui->tblImportProductReport->show();

QSqlQuery query;

query.prepare("SELECT SUM(`count`) FROM `practice\_3` "

"WHERE `direction` = 'i' AND `title` = ? "

"GROUP BY `title`");

query.addBindValue( title );

bool ok = query.exec();

bool exists = query.next();

(Продолжение кода на следующей странице)

(Продолжение кода)

if( ok&&exists ) {

ui->statusBar->clearMessage();

ui->outImportProductReportSummary->setText( query.value(0).toString() );

} else if( ok ) {

ui->statusBar->showMessage( tr("This product is not imported") );

} else {

ui->statusBar->showMessage( tr("Error occured while getting data") );

}

}

void MainWindow::on\_iptImportCountryReport\_returnPressed() {

QSqlError error = create\_import\_country\_report\_model();

if( error.isValid() ) {

QMessageBox::critical( this, tr("Error"), error.text() );

exit(1);

}

QString country = ui->iptImportCountryReport->text();

setup\_table\_view( ui->tblImportCountryReport );

update\_import\_country\_report();

ui->tblImportCountryReport->setEnabled( true );

ui->tblImportCountryReport->show();

QSqlQuery query;

query.prepare("SELECT SUM(`count`) FROM `practice\_3` "

"WHERE `direction` = 'i' AND `country` = ? "

"GROUP BY `country`");

query.addBindValue( country );

bool ok = query.exec();

bool exists = query.next();

if( ok && exists ) {

ui->statusBar->clearMessage();

ui->outImportCountryReportSummary->setText( query.value(0).toString() );

} else if( ok ) {

ui->statusBar->showMessage( tr("The products from this country were not imported") );

} else {

ui->statusBar->showMessage( tr("Error occured while getting data") );

}

}

void MainWindow::on\_iptGraphExport\_returnPressed() {

QString title = ui->iptGraphExport->text();

update\_graph\_export();

ui->wdgGraphExport->setEnabled( true );

if( !create\_chart( title ) ) {

ui->statusBar->showMessage("No data to display");

}

(Продолжение кода на следующей странице)

(Продолжение кода)

ui->wdgGraphExport->layout()->addWidget( chart\_view );

ui->wdgGraphExport->show();

}

/// Private functions

void MainWindow::update\_main\_overview() {

if( !create\_main\_overview\_model() ) {

QMessageBox::critical( this, tr("Error"), tr("Failed to create main data model") );

exit(1);

}

ui->tblMainOverview->setModel(main\_model);

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

ui->tblMainOverview->show();

ui->outExportLeadingCountry->clear();

QSqlQuery query;

query.prepare(

"SELECT `country` FROM `practice\_3` "

"WHERE `direction` = 'e' GROUP BY `country` "

"ORDER BY SUM(`count`) DESC LIMIT 1");

bool ok = query.exec();

bool exists = query.next();

if( ok && exists ) {

ui->statusBar->clearMessage();

ui->outExportLeadingCountry->setText( query.value(0).toString() );

} else if( ok ) {

ui->statusBar->showMessage( tr("No data to display") );

} else {

ui->statusBar->showMessage( tr("Error occured while getting data") );

}

}

void MainWindow::update\_import\_product\_report() {

ui->tblImportProductReport->setEnabled( false );

ui->outImportProductReportSummary->clear();

ui->iptImportProductReport->clear();

ui->iptImportProductReport->setFocus();

}

void MainWindow::update\_import\_country\_report() {

ui->tblImportCountryReport->setEnabled( false );

ui->outImportCountryReportSummary->clear();

ui->iptImportCountryReport->clear();

ui->iptImportCountryReport->setFocus();

}

(Продолжение кода на следующей странице)

(Продолжение кода)

void MainWindow::update\_graph\_export() {

ui->iptGraphExport->clear();

ui->iptGraphExport->setFocus();

ui->wdgGraphExport->setEnabled( false );

}

bool MainWindow::create\_main\_overview\_model() {

main\_model->setTable("practice\_3");

main\_model->setHeaderData( 0, Qt::Horizontal, tr("Id") );

main\_model->setHeaderData( 1, Qt::Horizontal, tr("Title") );

main\_model->setHeaderData( 2, Qt::Horizontal, tr("Amount") );

main\_model->setHeaderData( 3, Qt::Horizontal, tr("Year") );

main\_model->setHeaderData( 4, Qt::Horizontal, tr("Country") );

main\_model->setHeaderData( 5, Qt::Horizontal, tr("Direction") );

main\_model->setSort( 1, Qt::DescendingOrder );

main\_model->setEditStrategy( QSqlTableModel::OnFieldChange );

return main\_model->select();

}

QSqlError MainWindow::create\_import\_product\_report\_model() {

if( model ) {

delete model;

}

QString title = ui->iptImportProductReport->text();

QSqlQuery query;

query.prepare(

"SELECT `id`, `country`, SUM(`count`) AS summa "

"FROM `practice\_3` WHERE `direction` = 'i' AND `title` = ? "

"GROUP BY `country` ORDER BY summa DESC"

);

query.addBindValue( title );

query.exec();

model = new QSqlQueryModel( this );

model->setQuery( query );

model->setHeaderData( 0, Qt::Horizontal, tr("Id") );

model->setHeaderData( 1, Qt::Horizontal, tr("Country") );

model->setHeaderData( 2, Qt::Horizontal, tr("Summary amount") );

return model->lastError();

}

QSqlError MainWindow::create\_import\_country\_report\_model() {

if( model ) {

delete model;

}

QString country = ui->iptImportCountryReport->text();

QSqlQuery query;

query.prepare("SELECT `id`, `title`, SUM(`count`) FROM `practice\_3` "

"WHERE `direction` = 'i' AND `country` = ? "

"GROUP BY `title` ORDER BY SUM(`count`) DESC");

(Продолжение кода на следующей странице)

(Продолжение кода)

query.addBindValue( country );

query.exec();

model = new QSqlQueryModel( this );

model->setQuery( query );

model->setHeaderData( 0, Qt::Horizontal, tr("Id") );

model->setHeaderData( 1, Qt::Horizontal, tr("Product") );

model->setHeaderData( 2, Qt::Horizontal, tr("Summary amount") );

return model->lastError();

}

bool MainWindow::create\_chart( QString title ) {

// Dataset

if( chart\_set ) {

delete chart\_set;

}

chart\_set = new QBarSet( title, this );

// Categories

QStringList categories;

// Reading data from DB

QSqlQuery query;

query.prepare("SELECT `id`, `year`, SUM(`count`) "

"FROM `practice\_3` WHERE `title` = ? AND `direction` = 'e' "

"GROUP BY `year` ORDER BY `year`");

query.addBindValue( title );

query.exec();

bool exists = query.next();

if( !exists ) return false;

while( exists ) {

chart\_set->append( query.value(2).toInt() );

categories << query.value(1).toString();

exists = query.next();

}

// Series

if( chart\_series ) {

delete chart\_series;

}

chart\_series = new QBarSeries( this );

chart\_series->append( chart\_set );

// Chart

if( chart ) {

delete chart;

}

chart = new QChart();

chart->addSeries( chart\_series );

chart->setTitle( tr("Export of product by years") );

chart->setAnimationOptions( QChart::SeriesAnimations );

(Продолжение кода на следующей странице)

(Продолжение кода)

// Axis

if( chart\_axis ) {

delete chart\_axis;

}

chart\_axis = new QBarCategoryAxis( this );

chart\_axis->append( categories );

chart->createDefaultAxes();

chart->setAxisX( chart\_axis, chart\_series );

// Legend

chart->legend()->setVisible( true );

chart->legend()->setAlignment( Qt::AlignBottom );

ui->wdgGraphExport->layout()->removeWidget( chart\_view );

// View

if( chart\_view ) {

delete chart\_view;

}

chart\_view = new QChartView( chart, this );

chart\_view->setRenderHint( QPainter::Antialiasing );

chart\_view->setGeometry( 10, 40, 400, 350 );

chart\_view->setBaseSize( 400, 350 );

return true;

}

void MainWindow::setup\_table\_view( QTableView\* table\_view ) {

table\_view->setModel( model );

table\_view->setColumnHidden( 0, true );

table\_view->show();

}

bool MainWindow::setup\_database( QString dbname ) {

db.setDatabaseName( dbname );

db.open();

QSqlQuery query;

bool result = true;

if( !db.tables().contains( QLatin1String("practice\_3") ) ) {

// Create table

result = query.exec(

"CREATE TABLE `practice\_3` ("

" `id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,"

" `title` VARCHAR(255),"

" `count` INTEGER,"

" `year` YEAR,"

" `country` VARCHAR(255),"

" `direction` VARCHAR(1)"

")");

}

query.clear();

main\_model = new QSqlTableModel( this, db );

return result;

}

1. **Заголовочный файл модуля validator\_delegate.h [4]**

#ifndef DIRECTION\_VALIDATOR\_DELEGATE\_H

#define DIRECTION\_VALIDATOR\_DELEGATE\_H

#include <QItemDelegate>

/// Validate columns in main overview (delegate)

class ValidatorDelegate : public QItemDelegate

{

Q\_OBJECT

public:

explicit ValidatorDelegate(QObject \*parent = 0);

protected:

QWidget\* createEditor(QWidget \*parent, const QStyleOptionViewItem &option, const QModelIndex &index) const;

void setEditorData(QWidget \* editor, const QModelIndex & index) const;

void setModelData(QWidget \* editor, QAbstractItemModel \* model, const QModelIndex & index) const;

};

#endif // DIRECTION\_VALIDATOR\_DELEGATE\_H

1. **Исходный код модуля validator\_delegate.cpp [4]**

#include "validator\_delegate.h"

#include <QLineEdit>

#include <QRegExp>

#include <QRegExpValidator>

#include <QIntValidator>

ValidatorDelegate::ValidatorDelegate( QObject\* parent ) :

QItemDelegate( parent ) {}

QWidget\* ValidatorDelegate::createEditor( QWidget\* parent,

const QStyleOptionViewItem& option,

const QModelIndex& index ) const

{

QLineEdit\* editor = new QLineEdit( parent );

switch( index.column() ) {

case 1: // title

case 4: { // country

break;

}

case 2: { // count

editor->setValidator( new QIntValidator( 0, 16777215, editor ) );

break;

}

case 3: { // year

editor->setValidator( new QIntValidator( 1900, 2200, editor ) );

break;

}

case 5: { // direction

QRegExp direction\_re("export|import");

editor->setValidator( new QRegExpValidator( direction\_re ) );

break;

}

}

return editor;

}

void ValidatorDelegate::setEditorData( QWidget\* editor,

const QModelIndex& index ) const

{

QString value = index.model()->data( index, Qt::EditRole ).toString();

QLineEdit\* line = static\_cast<QLineEdit\*>( editor );

switch( index.column() ) {

case 1: // title

case 2: // count

case 3: // year

case 4: { // country

line->setText( value );

break;

}

(Продолжение кода на следующей странице)

(Продолжение кода)

case 5: { // direction

if( value == "i" ) {

line->setText("import");

} else {

line->setText("export");

}

break;

}

}

}

void ValidatorDelegate::setModelData( QWidget \*editor,

QAbstractItemModel \*model,

const QModelIndex &index ) const

{

QLineEdit \*line = static\_cast<QLineEdit\*>( editor );

QString value = line->text();

switch( index.column() ) {

case 1: // title

case 2: // count

case 3: // year

case 4: { // country

model->setData( index, value );

break;

}

case 5: { // direction

if( value == "import" ) {

model->setData( index, "i" );

} else {

model->setData( index, "e" );

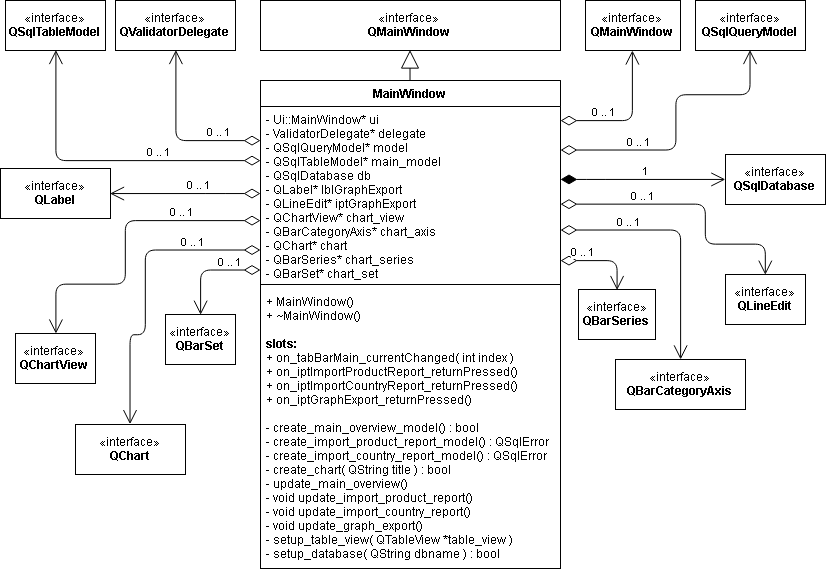
}

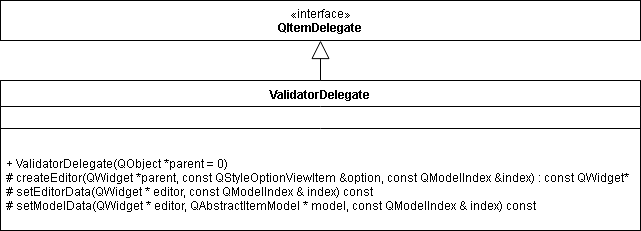
break;

}

}

}

1. **Диаграмма класса главного окна [1]**



1. **диаграмма класса валидатора [1]**