**南京航空航天大学**

《面向对象程序设计语言》课程设计报告

**家 电 销 售 管 理 系 统**

**学号：161310409**

**姓名：杜昊泽**

**日期：2014-5-31**

**目 录**

[一、需求分析 3](#_Toc389959329)

[二、程序的主要功能 3](#_Toc389959330)

[三、程序运行平台 3](#_Toc389959331)

[四、系统总框架图 4](#_Toc389959332)

[五、 程序类的说明 4](#_Toc389959333)

[六、模块分析 7](#_Toc389959334)

[七、比较有特色的函数 10](#_Toc389959335)

[八、存在的不足与对策、编程体会 14](#_Toc389959336)

[九、程序源代码 14](#_Toc389959337)

[**工程文件manager.pro** 14](#_Toc389959338)

[**头文件connection.h** 15](#_Toc389959339)

[**头文件logindialog.h** 17](#_Toc389959340)

[**头文件pieview.h** 17](#_Toc389959341)

[**头文件widget.h** 19](#_Toc389959342)

[**源文件logindialog.cpp** 22](#_Toc389959343)

[**源文件main.cpp** 23](#_Toc389959344)

[**源文件pieview.cpp** 24](#_Toc389959345)

[**源文件widget.cpp** 36](#_Toc389959346)

# 一、需求分析

由于社会生产力的发展以及人民购买力的提高，导致传统的商品销售记账方式不能胜任较密集的信息量的处理。因此，需要一个更加先进的工具来减少传统人工管理过程中工作人员的劳动强度，将劳动者从繁琐的数据处理中解放出来。

正因为C++是面向对象程序设计(Object Oriented Programming)的高级语言，并且保留了C语言的有效性、灵活性、便于移植的特点，提供了许多具有模板性质的库以及接口，所以有利于减少代码编写时所消耗的时间，更加适合于开发大中型的软件和应用程序。

Qt是一个基于C++的跨平台的应用程序和UI开发框架，具有很便捷的开发特性，利于节约开发时间。

# 二、程序的主要功能

1. 商品出售功能：

操作品牌表，读取该品牌单价，减小相应的库存量，增加相应的销售量，显示总金额。

1. 已有商品入库功能：

操作品牌表，增加相应品牌的进货量和库存量。

1. 新商品入库功能：

操作品牌表，增加新的品牌的名称、类型、单价、进货量、销售量和库存量。

1. 显示功能：

按时间顺序显示所有商品的购买记录。

1. 数据统计功能：

以表格形式按类别显示所有商品的购买量。

1. 构造数据图功能：

以图表形式按类别分别表示各品牌商品购买量与总购买量的关系。

1. 登陆界面及密码更改功能：

为保证软件内部数据的真实性与安全性，需对使用者身份以密码形式进行校验，并支持登陆后修改密码，初始密码为123456。

1. 关于页面：

显示程序及作者信息。

# 三、程序运行平台

1. 本程序是通过Qt Creator 3.0.1(Based on Qt 5.2.1 (MSVC 2010, 32 bit))编写的，可以在IDE的开发环境中运行。

具体操作如下：打开Qt Creator 3.0.1，在欢迎界面点击“Open Project”，找到“manager.pro”工程文件，点击“运行”命令运行该程序。

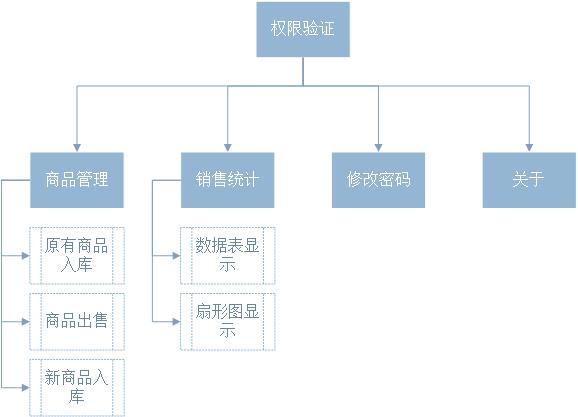
1. 本程序已发布，可以在基于Windows NT 5.x（Windows XP）及以上环境中运行（测试用真实环境为Windows 8.1，利用虚拟机Virtual Box在Windows XP环境通过测试）。

具体操作如下：打开应用程序所在的文件夹（build-manager-桌面-Release），双击“manager.exe”运行。

1. 本程序已发布，可以在Linux环境中运行（测试用环境为Ubuntu 14.04 desktop LTS，内核版本为Linux 3.13.0-19-generic）。

具体操作如下：打开应用程序所在目录（build-manager-桌面-Release）双击“manager”运行。

# 四、系统总框架图



# 程序类的说明

1. 主窗体类

//本类完成主窗体中各控件的声明

class Widget : public QWidget

{

Q\_OBJECT

public:

explicit Widget(QWidget \*parent = 0);

~*Widget*();

//由于购买记录以时间为标志，故以下为使用Qt内部时间获取的声明

enum DateTimeType{Time, Date, DateTime};

QString getDateTime(DateTimeType type);

//以下完成对主窗体中各控件槽与信号的声明

private slots:

void on\_sellTypeComboBox\_currentIndexChanged(QString );

void on\_sellBrandComboBox\_currentIndexChanged(QString );

void on\_sellNumSpinBox\_valueChanged(int );

void on\_sellCancelBtn\_clicked();

void on\_sellOkBtn\_clicked();

void on\_typeComboBox\_currentIndexChanged(QString );

void on\_updateBtn\_clicked();

void on\_manageBtn\_clicked();

void on\_chartBtn\_clicked();

void on\_changePwdBtn\_clicked();

void on\_passwordBtn\_clicked();

void on\_goodsTypeComboBox\_currentIndexChanged(QString );

void on\_goodsCancelBtn\_clicked();

void on\_goodsBrandComboBox\_currentIndexChanged(QString );

void on\_goodsNumSpinBox\_valueChanged(int );

void on\_goodsOkBtn\_clicked();

void on\_newTypeComboBox\_currentIndexChanged(QString );

void on\_newCancelBtn\_clicked();

void on\_newNumSpinBox\_valueChanged(int );

void on\_newPriceSpinBox\_valueChanged(int );

void on\_newBrandLineEdit\_textChanged(QString );

void on\_newOkBtn\_clicked();

void on\_aboutBtn\_clicked();

//以下声明对界面文件的调用、对xml的读写和绘图界面的构建

private:

Ui::Widget \*ui;

QDomDocument doc;

QStandardItemModel \*chartModel;

//以下声明对xml的读写

private:

bool docRead();

bool docWrite();

void writeXml();

void createNodes(QDomElement &date); //以时间创建记录节点的声明

void showDailyList();

void createChartModelView();

void showChart();

};

1. 登录窗体类

//本类完成对登陆界面的声明

class LoginDialog : public QDialog

{

Q\_OBJECT

//以下声明登陆界面的继承关系及后续操作（显示主界面或关闭）

public:

explicit LoginDialog(QWidget \*parent = 0);

~*LoginDialog*();

//以下完成对登陆界面中“确定”与“取消”按钮槽与信号的声明

private slots:

void on\_loginBtn\_clicked();//“确定”信号的声明

void on\_quitBtn\_clicked();//“取消”信号的声明

//以下声明对界面文件的调用

private:

Ui::LoginDialog \*ui;

};

1. 数据统计及绘图界面类

//本类完成对绘图界面的声明

class PieView : public QAbstractItemView

{

Q\_OBJECT

public:

PieView(QWidget \*parent = 0);

QRect *visualRect*(const QModelIndex &index) const;

void *scrollTo*(const QModelIndex &index, ScrollHint hint = EnsureVisible);

QModelIndex *indexAt*(const QPoint &point) const;

protected slots:

void *dataChanged*(const QModelIndex &topLeft, const QModelIndex &bottomRight);

void *rowsInserted*(const QModelIndex &parent, int start, int end);

void *rowsAboutToBeRemoved*(const QModelIndex &parent, int start, int end);

protected:

bool *edit*(const QModelIndex &index, EditTrigger trigger, QEvent \*event);

QModelIndex *moveCursor*(QAbstractItemView::CursorAction cursorAction, Qt::KeyboardModifiers modifiers);

int *horizontalOffset*() const;

int *verticalOffset*() const;

bool *isIndexHidden*(const QModelIndex &index) const;

void *setSelection*(const QRect&, QItemSelectionModel::SelectionFlags command);

void *mousePressEvent*(QMouseEvent \*event);

void *mouseMoveEvent*(QMouseEvent \*event);

void *mouseReleaseEvent*(QMouseEvent \*event);

void *paintEvent*(QPaintEvent \*event);

void *resizeEvent*(QResizeEvent \*event);

void *scrollContentsBy*(int dx, int dy);

QRegion *visualRegionForSelection*(const QItemSelection &selection) const;

private:

QRect itemRect(const QModelIndex &item) const;

QRegion itemRegion(const QModelIndex &index) const;

int rows(const QModelIndex &index = QModelIndex()) const;

void *updateGeometries*();

int margin;

int totalSize;

int pieSize;

int validItems;

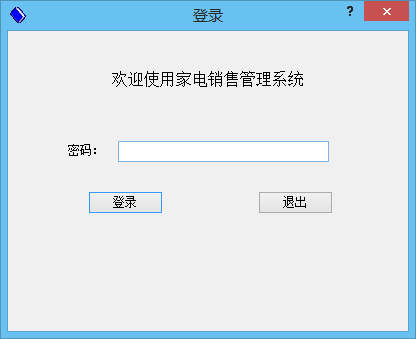
double totalValue;

QPoint origin;

QRubberBand \*rubberBand;

};

# 六、模块分析

1. 用户界面登录模块

系统要求用户输入密码，并将密码与数据库data.db内password表中记录比对，如果结果正确则进入主窗体，否则要求用户继续输入密码。

1. 商品出售模块

由用户输入商品信息，完成销售操作，修改数据库data.db内brand表中的相应品牌的库存量和销售量。

1. 原有商品入库模块

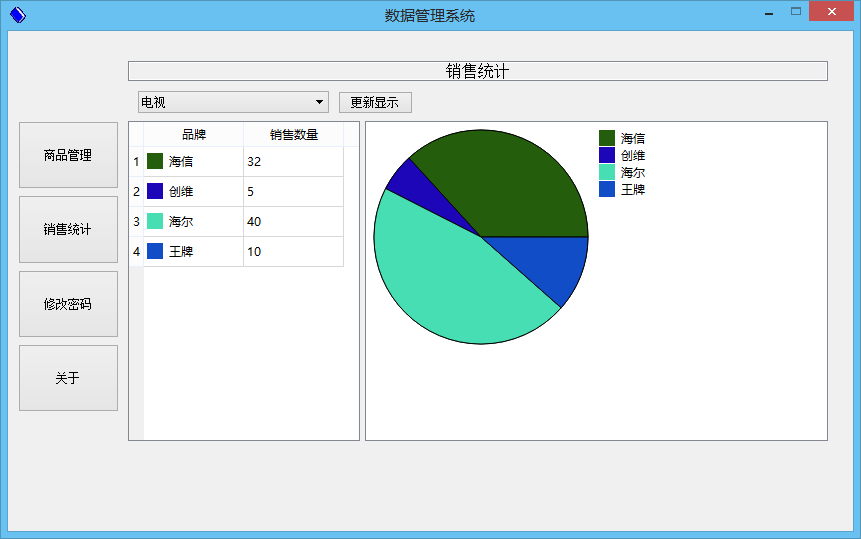
由用户输入商品信息，完成原有商品入库操作，修改数据库data.db内brand表中的相应品牌的库存量和进货量。

1. 新商品入库模块

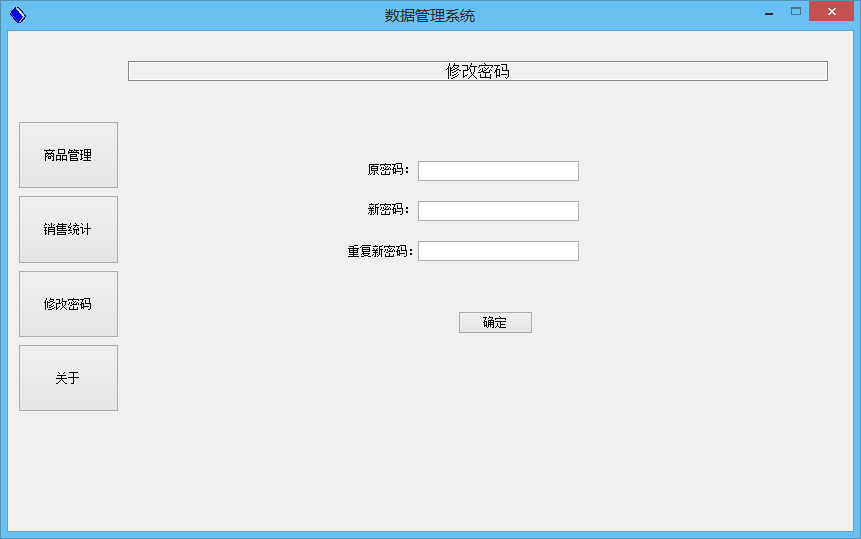
由用户输入新商品信息，完成新商品入库操作，在数据库data.db内brand表中添加新品牌的记录。

1. 显示及数据记录模块

每次完成购买操作时，将购买信息输出至data.xml，并读取显示在主窗体相应位置。

1. 销售统计数据表及数据图显示模块

读取数据库data.db内销售总量和各记录的销售量，输出至数据表中，并在绘图区绘制扇形图。

1. 修改密码模块

系统要求用户输入密码，并将密码与数据库data.db内password表中记录比对，如果结果正确则比对新密码与重复是否一致并完成对数据库data.db内password表的修改。

1. 关于模块

显示程序及作者信息。

# 七、比较有特色的函数

1. 数据库data.db的构造

//声明返回值为布尔型有利于直接对是否成功构造数据库进行判断

static bool createConnection()

{

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

db.setHostName("duhaoze1995");

db.setDatabaseName("data.db");

db.setUserName("dahaoze");

db.setPassword("123456");

if (!db.open()) {

//提示出错

return false;

}

QSqlQuery query;

// 创建分类表

query.exec("create table type(id varchar primary key, name varchar)");

query.exec(QString("insert into type values('0', '请选择类型')"));

query.exec(QString("insert into type values('01', '电视')"));

query.exec(QString("insert into type values('02', '空调')"));

// 创建品牌表

query.exec("create table brand(id varchar primary key, name varchar, "

"type varchar, price int, sum int, sell int, last int)");

query.exec(QString("insert into brand values('01', '海信', '电视', 3699, 50, 10, 40)"));

// 此处省略一些样例数据

// 创建密码表

query.exec("create table password(pwd varchar primary key)");

query.exec("insert into password values('123456')");

return true;

}

1. 日销售清单data.xml的构造

static bool createXml()

{

QFile file("data.xml");

if(file.exists()) return true; // 如果已存在此文件则不重复创建

if(!file.*open*(QIODevice::WriteOnly | QIODevice::Truncate)) return false; // 如果文件无法打开则返回错误

QDomDocument doc;

QDomProcessingInstruction instruction;

instruction = doc.createProcessingInstruction("xml","version=\"1.0\" encoding=\"UTF-8\"");

doc.appendChild(instruction);

QDomElement root = doc.createElement(QString("日销售清单"));//创建根节点

doc.appendChild(root); //在文件中添加根节点

QTextStream out(&file); //输出到文件

doc.save(out,4);

file.*close*();

return true;

}

1. 创建日销售记录节点

// 创建节点

void Widget::createNodes(QDomElement &date)

{

QDomElement time = doc.createElement(QString("时间"));

QDomAttr curTime = doc.createAttribute("time");

curTime.setValue(getDateTime(Time));

time.setAttributeNode(curTime);

date.appendChild(time); //以时间创建子节点挂在根节点下

//以下定义新的子节点存储购买的详细信息

QDomElement type = doc.createElement(QString("类型"));

QDomElement brand = doc.createElement(QString("品牌"));

QDomElement price = doc.createElement(QString("单价"));

QDomElement num = doc.createElement(QString("数量"));

QDomElement sum = doc.createElement(QString("金额"));

QDomText text;

text = doc.createTextNode(QString("%1")

.arg(ui->sellTypeComboBox->currentText()));

type.appendChild(text);

text = doc.createTextNode(QString("%1")

.arg(ui->sellBrandComboBox->currentText()));

brand.appendChild(text);

text = doc.createTextNode(QString("%1")

.arg(ui->sellPriceLineEdit->text()));

price.appendChild(text);

text = doc.createTextNode(QString("%1")

.arg(ui->sellNumSpinBox->value()));

num.appendChild(text);

text = doc.createTextNode(QString("%1")

.arg(ui->sellSumLineEdit->text()));

sum.appendChild(text);

time.appendChild(type);

time.appendChild(brand);

time.appendChild(price);

time.appendChild(num);

time.appendChild(sum);

}

1. 销售记录图表相关的函数

// 创建销售记录图表的模型和视图

void Widget::createChartModelView()

{

chartModel = new QStandardItemModel(this);

chartModel->setColumnCount(2);

chartModel->*setHeaderData*(0, Qt::Horizontal, QString("品牌"));

chartModel->*setHeaderData*(1, Qt::Horizontal, QString("销售数量"));

//在绘图区中绘图

QSplitter \*splitter = new QSplitter(ui->chartPage);

splitter->resize(700, 320);

splitter->move(0, 80);

QTableView \*table = new QTableView;

PieView \*pieChart = new PieView;

splitter->addWidget(table);

splitter->addWidget(pieChart);

splitter->setStretchFactor(0, 1);

splitter->setStretchFactor(1, 2);

table->*setModel*(chartModel);

pieChart->*setModel*(chartModel);

QItemSelectionModel \*selectionModel = new QItemSelectionModel(chartModel);

table->*setSelectionModel*(selectionModel);

pieChart->*setSelectionModel*(selectionModel);

}

// 显示销售记录图表

void Widget::showChart()

{

QSqlQuery query;

//选择类型

query.exec(QString("select name,sell from brand where type='%1'")

.arg(ui->typeComboBox->currentText()));

chartModel->*removeRows*(0, chartModel->*rowCount*(QModelIndex()), QModelIndex());

int row = 0;

while(query.next()) {

//随机产生各扇形区域的颜色

int r = qrand() % 256;

int g = qrand() % 256;

int b = qrand() % 256;

chartModel->*insertRows*(row, 1, QModelIndex());

//读取品牌表

chartModel->*setData*(chartModel->*index*(row, 0, QModelIndex()),

query.value(0).toString());

//读取销售量

chartModel->*setData*(chartModel->*index*(row, 1, QModelIndex()),

query.value(1).toInt());

//设置颜色

chartModel->*setData*(chartModel->*index*(row, 0, QModelIndex()),

QColor(r, g, b), Qt::DecorationRole);

row++;

}

}

# 八、存在的不足与对策、编程体会

由于时间较为仓促，没有系统地学习SQL数据库及xml的知识，程序中用到的SQL数据库及xml的相关操作的应用都出于霍亚飞所著《Qt Creator快速入门》一书以及网络搜索结果。

对于绘扇形图使用的是网络上的模型，结合从数据库读出的数据进行绘图。但自己对颜色上加入了随机处理操作。

不足之处还有界面比较粗糙、不美观，没有更多地体现数据结构和算法的思想。

下一步我会进行比较系统的SQL数据库和xml文件的学习，彻底弄明白中间每一行代码的作用，然后尝试做出更多的基于SQL数据库和xml的应用程序。

通过本次课程设计的编程体验，我第一次感受了一个真正的窗体程序的编写过程。它和终端界面的程序有一些区别，更和以前所做的主要是为了解决特定的数学问题而编写的终端界面的ACM赛题程序有区别。这次的经历会使我了解图形化和GUI设计的一些知识，提高程序工程的开发能力。

# 九、程序源代码

## **工程文件manager.pro**

QT += core gui

QT += sql xml

TARGET = manager

TEMPLATE = app

SOURCES += main.cpp\

widget.cpp \

pieview.cpp \

logindialog.cpp

HEADERS += widget.h \

connection.h \

pieview.h \

logindialog.h

FORMS += widget.ui \

logindialog.ui

OTHER\_FILES += \

my.rc

RC\_FILE = \

my.rc

## **头文件connection.h**

#ifndef CONNECTION\_H

#define CONNECTION\_H

#include <QtSql>

#include <QDebug>

#include <QtXml>

static bool createConnection()

{

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

db.setHostName("duhaoze1995");

db.setDatabaseName("data.db");

db.setUserName("dahaoze1995");

db.setPassword("123456");

if (!db.open()) {

//提示出错

return false;

}

QSqlQuery query;

// 创建分类表

query.exec("create table type(id varchar primary key, name varchar)");

query.exec(QString("insert into type values('0', '请选择类型')"));

query.exec(QString("insert into type values('01', '电视')"));

query.exec(QString("insert into type values('02', '空调')"));

// 创建品牌表

query.exec("create table brand(id varchar primary key, name varchar, "

"type varchar, price int, sum int, sell int, last int)");

query.exec(QString("insert into brand values('01', '海信', '电视', 3699, 50, 10, 40)"));

query.exec(QString("insert into brand values('02', '创维', '电视', 3499, 20, 5, 15)"));

query.exec(QString("insert into brand values('03', '海尔', '电视', 4199, 80, 40, 40)"));

query.exec(QString("insert into brand values('04', '王牌', '电视', 3999, 40, 10, 30)"));

query.exec(QString("insert into brand values('05', '海尔', '空调', 2899, 60, 10, 50)"));

query.exec(QString("insert into brand values('06', '格力', '空调', 2799, 70, 20, 50)"));

// 创建密码表

query.exec("create table password(pwd varchar primary key)");

query.exec("insert into password values('123456')");

return true;

}

static bool createXml()

{

QFile file("data.xml");

if(file.exists()) return true;

if(!file.*open*(QIODevice::WriteOnly | QIODevice::Truncate)) return false;

QDomDocument doc;

QDomProcessingInstruction instruction;

instruction = doc.createProcessingInstruction("xml","version=\"1.0\" encoding=\"UTF-8\"");

doc.appendChild(instruction);

QDomElement root = doc.createElement(QString("日销售清单"));

doc.appendChild(root);

QTextStream out(&file);

doc.save(out,4);

file.*close*();

return true;

}

#endif // CONNECTION\_H

## **头文件logindialog.h**

#ifndef LOGINDIALOG\_H

#define LOGINDIALOG\_H

#include <QDialog>

namespace Ui {

class LoginDialog;

}

class LoginDialog : public QDialog

{

Q\_OBJECT

public:

explicit LoginDialog(QWidget \*parent = 0);

~*LoginDialog*();

private slots:

void on\_loginBtn\_clicked();

void on\_quitBtn\_clicked();

private:

Ui::LoginDialog \*ui;

};

#endif // LOGINDIALOG\_H

## **头文件pieview.h**

#ifndef PIEVIEW\_H

#define PIEVIEW\_H

#include <QAbstractItemView>

#include <QFont>

#include <QItemSelection>

#include <QItemSelectionModel>

#include <QModelIndex>

#include <QRect>

#include <QSize>

#include <QPoint>

#include <QWidget>

QT\_BEGIN\_NAMESPACE

class QRubberBand;

QT\_END\_NAMESPACE

class PieView : public QAbstractItemView

{

Q\_OBJECT//

public:

PieView(QWidget \*parent = 0);

QRect *visualRect*(const QModelIndex &index) const;

void *scrollTo*(const QModelIndex &index, ScrollHint hint = EnsureVisible);

QModelIndex *indexAt*(const QPoint &point) const;

protected slots:

void *dataChanged*(const QModelIndex &topLeft, const QModelIndex &bottomRight);

void *rowsInserted*(const QModelIndex &parent, int start, int end);

void *rowsAboutToBeRemoved*(const QModelIndex &parent, int start, int end);

protected:

bool *edit*(const QModelIndex &index, EditTrigger trigger, QEvent \*event);

QModelIndex *moveCursor*(QAbstractItemView::CursorAction cursorAction,

Qt::KeyboardModifiers modifiers);

int *horizontalOffset*() const;

int *verticalOffset*() const;

bool *isIndexHidden*(const QModelIndex &index) const;

void *setSelection*(const QRect&, QItemSelectionModel::SelectionFlags command);

void *mousePressEvent*(QMouseEvent \*event);

void *mouseMoveEvent*(QMouseEvent \*event);

void *mouseReleaseEvent*(QMouseEvent \*event);

void *paintEvent*(QPaintEvent \*event);

void *resizeEvent*(QResizeEvent \*event);

void *scrollContentsBy*(int dx, int dy);

QRegion *visualRegionForSelection*(const QItemSelection &selection) const;

private:

QRect itemRect(const QModelIndex &item) const;

QRegion itemRegion(const QModelIndex &index) const;

int rows(const QModelIndex &index = QModelIndex()) const;

void *updateGeometries*();

int margin;

int totalSize;

int pieSize;

int validItems;

double totalValue;

QPoint origin;

QRubberBand \*rubberBand;

};

#endif

## **头文件widget.h**

#ifndef WIDGET\_H

#define WIDGET\_H

#include <QWidget>

#include <QDomDocument>

class QStandardItemModel;

namespace Ui {

class Widget;

}

class Widget : public QWidget

{

Q\_OBJECT

public:

explicit Widget(QWidget \*parent = 0);

~*Widget*();

enum DateTimeType{Time, Date, DateTime};

QString getDateTime(DateTimeType type);

private slots:

void on\_sellTypeComboBox\_currentIndexChanged(QString );

void on\_sellBrandComboBox\_currentIndexChanged(QString );

void on\_sellNumSpinBox\_valueChanged(int );

void on\_sellCancelBtn\_clicked();

void on\_sellOkBtn\_clicked();

void on\_typeComboBox\_currentIndexChanged(QString );

void on\_updateBtn\_clicked();

void on\_manageBtn\_clicked();

void on\_chartBtn\_clicked();

void on\_changePwdBtn\_clicked();

void on\_passwordBtn\_clicked();

void on\_goodsTypeComboBox\_currentIndexChanged(QString );

void on\_goodsCancelBtn\_clicked();

void on\_goodsBrandComboBox\_currentIndexChanged(QString );

void on\_goodsNumSpinBox\_valueChanged(int );

void on\_goodsOkBtn\_clicked();

void on\_newTypeComboBox\_currentIndexChanged(QString );

void on\_newCancelBtn\_clicked();

void on\_newNumSpinBox\_valueChanged(int );

void on\_newPriceSpinBox\_valueChanged(int );

void on\_newBrandLineEdit\_textChanged(QString );

void on\_newOkBtn\_clicked();

void on\_aboutBtn\_clicked();

private:

Ui::Widget \*ui;

QDomDocument doc;

QStandardItemModel \*chartModel;

private:

bool docRead();

bool docWrite();

void writeXml();

void createNodes(QDomElement &date);

void showDailyList();

void createChartModelView();

void showChart();

};

#endif // WIDGET\_H

## **源文件logindialog.cpp**

#include "logindialog.h"

#include "ui\_logindialog.h"

#include <QMessageBox>

#include <QSqlQuery>

#include <QDebug>

LoginDialog::LoginDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::LoginDialog)

{

ui->setupUi(this);

setFixedSize(400, 300);

setWindowTitle(tr("登录"));

ui->pwdLineEdit->setFocus();

ui->loginBtn->setDefault(true);

}

LoginDialog::~*LoginDialog*()

{

delete ui;

}

// 登录按钮

void LoginDialog::on\_loginBtn\_clicked()

{

if (ui->pwdLineEdit->text().isEmpty()) {

QMessageBox::information(this, tr("请输入密码"),

tr("请先输入密码再登录！"), QMessageBox::Ok);

ui->pwdLineEdit->setFocus();

} else {

QSqlQuery query;

query.exec("select pwd from password");

query.next();

if (query.value(0).toString() == ui->pwdLineEdit->text()) {

QDialog::*accept*();

} else {

QMessageBox::warning(this, tr("密码错误"),

tr("请输入正确的密码再登录！"), QMessageBox::Ok);

ui->pwdLineEdit->clear();

ui->pwdLineEdit->setFocus();

}

}

}

// 退出按钮

void LoginDialog::on\_quitBtn\_clicked()

{

QDialog::*reject*();

}

## **源文件main.cpp**

#include <QtGui>

#include <QApplication>

#include "widget.h"

#include <QTextCodec>

#include "connection.h"

#include "logindialog.h"

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

{

QApplication a(argc, argv);

// 必须在创建数据库之前使用，不然无法在数据库中使用中文

QTextCodec::setCodecForTr(QTextCodec::codecForLocale());

QTextCodec::setCodecForCStrings(QTextCodec::codecForLocale());

if(!createConnection() || !createXml()) return 0;

Widget w;

LoginDialog dlg;

if (dlg.exec() == QDialog::Accepted) {

w.show();

return a.exec();

} else {

return 0;

}

}

## **源文件pieview.cpp**

#include <math.h>

#include <QtGui>

#ifndef M\_PI

#define M\_PI 3.1415927

#endif

#include "pieview.h"

PieView::PieView(QWidget \*parent)

: QAbstractItemView(parent)

{

horizontalScrollBar()->setRange(0, 0);

verticalScrollBar()->setRange(0, 0);

margin = 8;

totalSize = 230;

pieSize = totalSize - 2\*margin;

validItems = 0;

totalValue = 0.0;

rubberBand = 0;

}

void PieView::*dataChanged*(const QModelIndex &topLeft,

const QModelIndex &bottomRight)

{

QAbstractItemView::*dataChanged*(topLeft, bottomRight);

validItems = 0;

totalValue = 0.0;

for (int row = 0; row < model()->*rowCount*(rootIndex()); ++row) {

QModelIndex index = model()->*index*(row, 1, rootIndex());

double value = model()->*data*(index).toDouble();

if (value > 0.0) {

totalValue += value;

validItems++;

}

}

viewport()->update();

}

bool PieView::*edit*(const QModelIndex &index, EditTrigger trigger, QEvent \*event)

{

if (index.column() == 0)

return QAbstractItemView::*edit*(index, trigger, event);

else

return false;

}

QModelIndex PieView::*indexAt*(const QPoint &point) const

{

if (validItems == 0)

return QModelIndex();

int wx = point.x() + horizontalScrollBar()->value();

int wy = point.y() + verticalScrollBar()->value();

if (wx < totalSize) {

double cx = wx - totalSize/2;

double cy = totalSize/2 - wy;

double d = pow(pow(cx, 2) + pow(cy, 2), 0.5);

if (d == 0 || d > pieSize/2)

return QModelIndex();

double angle = (180 / M\_PI) \* acos(cx/d);

if (cy < 0)

angle = 360 - angle;

double startAngle = 0.0;

for (int row = 0; row < model()->*rowCount*(rootIndex()); ++row) {

QModelIndex index = model()->*index*(row, 1, rootIndex());

double value = model()->*data*(index).toDouble();

if (value > 0.0) {

double sliceAngle = 360\*value/totalValue;

if (angle >= startAngle && angle < (startAngle + sliceAngle))

return model()->*index*(row, 1, rootIndex());

startAngle += sliceAngle;

}

}

} else {

double itemHeight = QFontMetrics(*viewOptions*().font).height();

int listItem = int((wy - margin) / itemHeight);

int validRow = 0;

for (int row = 0; row < model()->*rowCount*(rootIndex()); ++row) {

QModelIndex index = model()->*index*(row, 1, rootIndex());

if (model()->*data*(index).toDouble() > 0.0) {

if (listItem == validRow)

return model()->*index*(row, 0, rootIndex());

validRow++;

}

}

}

return QModelIndex();

}

bool PieView::*isIndexHidden*(const QModelIndex & /\*index\*/) const

{

return false;

}

QRect PieView::itemRect(const QModelIndex &index) const

{

if (!index.isValid())

return QRect();

QModelIndex valueIndex;

if (index.column() != 1)

valueIndex = model()->*index*(index.row(), 1, rootIndex());

else

valueIndex = index;

if (model()->*data*(valueIndex).toDouble() > 0.0) {

int listItem = 0;

for (int row = index.row()-1; row >= 0; --row) {

if (model()->*data*(model()->*index*(row, 1, rootIndex())).toDouble() > 0.0)

listItem++;

}

double itemHeight;

switch (index.column()) {

case 0:

itemHeight = QFontMetrics(*viewOptions*().font).height();

return QRect(totalSize,

int(margin + listItem\*itemHeight),

totalSize - margin, int(itemHeight));

case 1:

return viewport()->rect();

}

}

return QRect();

}

QRegion PieView::itemRegion(const QModelIndex &index) const

{

if (!index.isValid())

return QRegion();

if (index.column() != 1)

return itemRect(index);

if (model()->*data*(index).toDouble() <= 0.0)

return QRegion();

double startAngle = 0.0;

for (int row = 0; row < model()->*rowCount*(rootIndex()); ++row) {

QModelIndex sliceIndex = model()->*index*(row, 1, rootIndex());

double value = model()->*data*(sliceIndex).toDouble();

if (value > 0.0) {

double angle = 360\*value/totalValue;

if (sliceIndex == index) {

QPainterPath slicePath;

slicePath.moveTo(totalSize/2, totalSize/2);

slicePath.arcTo(margin, margin, margin+pieSize, margin+pieSize,

startAngle, angle);

slicePath.closeSubpath();

return QRegion(slicePath.toFillPolygon().toPolygon());

}

startAngle += angle;

}

}

return QRegion();

}

int PieView::*horizontalOffset*() const

{

return horizontalScrollBar()->value();

}

void PieView::*mousePressEvent*(QMouseEvent \*event)

{

QAbstractItemView::*mousePressEvent*(event);

origin = event->pos();

if (!rubberBand)

rubberBand = new QRubberBand(QRubberBand::Rectangle, viewport());

rubberBand->setGeometry(QRect(origin, QSize()));

rubberBand->show();

}

void PieView::*mouseMoveEvent*(QMouseEvent \*event)

{

if (rubberBand)

rubberBand->setGeometry(QRect(origin, event->pos()).normalized());

QAbstractItemView::*mouseMoveEvent*(event);

}

void PieView::*mouseReleaseEvent*(QMouseEvent \*event)

{

QAbstractItemView::*mouseReleaseEvent*(event);

if (rubberBand)

rubberBand->hide();

viewport()->update();

}

QModelIndex PieView::*moveCursor*(QAbstractItemView::CursorAction cursorAction,

Qt::KeyboardModifiers /\*modifiers\*/)

{

QModelIndex current = currentIndex();

switch (cursorAction) {

case MoveLeft:

case MoveUp:

if (current.row() > 0)

current = model()->*index*(current.row() - 1, current.column(),

rootIndex());

else

current = model()->*index*(0, current.column(), rootIndex());

break;

case MoveRight:

case MoveDown:

if (current.row() < rows(current) - 1)

current = model()->*index*(current.row() + 1, current.column(),

rootIndex());

else

current = model()->*index*(rows(current) - 1, current.column(),

rootIndex());

break;

default:

break;

}

viewport()->update();

return current;

}

void PieView::*paintEvent*(QPaintEvent \*event)

{

QItemSelectionModel \*selections = selectionModel();

QStyleOptionViewItem option = *viewOptions*();

QBrush background = option.palette.base();

QPen foreground(option.palette.color(QPalette::WindowText));

QPainter painter(viewport());

painter.setRenderHint(QPainter::Antialiasing);

painter.fillRect(event->rect(), background);

painter.setPen(foreground);

QRect pieRect = QRect(margin, margin, pieSize, pieSize);

if (validItems > 0) {

// 绘制圆形饼状图

painter.save();

painter.translate(pieRect.x() - horizontalScrollBar()->value(),

pieRect.y() - verticalScrollBar()->value());

painter.drawEllipse(0, 0, pieSize, pieSize);

double startAngle = 0.0;

int row;

for (row = 0; row < model()->*rowCount*(rootIndex()); ++row) {

QModelIndex index = model()->*index*(row, 1, rootIndex());

double value = model()->*data*(index).toDouble();

if (value > 0.0) {

double angle = 360\*value/totalValue;

QModelIndex colorIndex = model()->*index*(row, 0, rootIndex());

QColor color = QColor(model()->*data*(colorIndex,

Qt::DecorationRole).toString());

if (currentIndex() == index)

painter.setBrush(QBrush(color, Qt::Dense4Pattern));

else if (selections->isSelected(index))

painter.setBrush(QBrush(color, Qt::Dense3Pattern));

else

painter.setBrush(QBrush(color));

painter.drawPie(0, 0, pieSize, pieSize, int(startAngle\*16),

int(angle\*16));

startAngle += angle;

}

}

painter.restore();

// 绘制饼状图旁边的图示

int keyNumber = 0;

for (row = 0; row < model()->*rowCount*(rootIndex()); ++row) {

QModelIndex index = model()->*index*(row, 1, rootIndex());

double value = model()->*data*(index).toDouble();

if (value > 0.0) {

QModelIndex labelIndex = model()->*index*(row, 0, rootIndex());

QStyleOptionViewItem option = *viewOptions*();

option.rect = *visualRect*(labelIndex);

if (selections->isSelected(labelIndex))

option.state |= QStyle::State\_Selected;

if (currentIndex() == labelIndex)

option.state |= QStyle::State\_HasFocus;

itemDelegate()->*paint*(&painter, option, labelIndex);

keyNumber++;

}

}

}

}

void PieView::*resizeEvent*(QResizeEvent \* /\* event \*/)

{

*updateGeometries*();

}

int PieView::rows(const QModelIndex &index) const

{

return model()->*rowCount*(model()->*parent*(index));

}

void PieView::*rowsInserted*(const QModelIndex &parent, int start, int end)

{

for (int row = start; row <= end; ++row) {

QModelIndex index = model()->*index*(row, 1, rootIndex());

double value = model()->*data*(index).toDouble();

if (value > 0.0) {

totalValue += value;

validItems++;

}

}

QAbstractItemView::*rowsInserted*(parent, start, end);

}

void PieView::*rowsAboutToBeRemoved*(const QModelIndex &parent, int start, int end)

{

for (int row = start; row <= end; ++row) {

QModelIndex index = model()->*index*(row, 1, rootIndex());

double value = model()->*data*(index).toDouble();

if (value > 0.0) {

totalValue -= value;

validItems--;

}

}

QAbstractItemView::*rowsAboutToBeRemoved*(parent, start, end);

}

void PieView::*scrollContentsBy*(int dx, int dy)

{

viewport()->scroll(dx, dy);

}

void PieView::*scrollTo*(const QModelIndex &index, ScrollHint)

{

QRect area = viewport()->rect();

QRect rect = *visualRect*(index);

if (rect.left() < area.left())

horizontalScrollBar()->setValue(

horizontalScrollBar()->value() + rect.left() - area.left());

else if (rect.right() > area.right())

horizontalScrollBar()->setValue(

horizontalScrollBar()->value() + qMin(

rect.right() - area.right(), rect.left() - area.left()));

if (rect.top() < area.top())

verticalScrollBar()->setValue(

verticalScrollBar()->value() + rect.top() - area.top());

else if (rect.bottom() > area.bottom())

verticalScrollBar()->setValue(

verticalScrollBar()->value() + qMin(

rect.bottom() - area.bottom(), rect.top() - area.top()));

update();

}

void PieView::*setSelection*(const QRect &rect, QItemSelectionModel::SelectionFlags command)

{

QRect contentsRect = rect.translated(

horizontalScrollBar()->value(),

verticalScrollBar()->value()).normalized();

int rows = model()->*rowCount*(rootIndex());

int columns = model()->*columnCount*(rootIndex());

QModelIndexList indexes;

for (int row = 0; row < rows; ++row) {

for (int column = 0; column < columns; ++column) {

QModelIndex index = model()->*index*(row, column, rootIndex());

QRegion region = itemRegion(index);

if (!region.intersect(contentsRect).isEmpty())

indexes.append(index);

}

}

if (indexes.size() > 0) {

int firstRow = indexes[0].row();

int lastRow = indexes[0].row();

int firstColumn = indexes[0].column();

int lastColumn = indexes[0].column();

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

firstRow = qMin(firstRow, indexes[i].row());

lastRow = qMax(lastRow, indexes[i].row());

firstColumn = qMin(firstColumn, indexes[i].column());

lastColumn = qMax(lastColumn, indexes[i].column());

}

QItemSelection selection(

model()->*index*(firstRow, firstColumn, rootIndex()),

model()->*index*(lastRow, lastColumn, rootIndex()));

selectionModel()->*select*(selection, command);

} else {

QModelIndex noIndex;

QItemSelection selection(noIndex, noIndex);

selectionModel()->*select*(selection, command);

}

update();

}

void PieView::*updateGeometries*()

{

horizontalScrollBar()->setPageStep(viewport()->width());

horizontalScrollBar()->setRange(0, qMax(0, 2\*totalSize - viewport()->width()));

verticalScrollBar()->setPageStep(viewport()->height());

verticalScrollBar()->setRange(0, qMax(0, totalSize - viewport()->height()));

}

int PieView::*verticalOffset*() const

{

return verticalScrollBar()->value();

}

QRect PieView::*visualRect*(const QModelIndex &index) const

{

QRect rect = itemRect(index);

if (rect.isValid())

return QRect(rect.left() - horizontalScrollBar()->value(),

rect.top() - verticalScrollBar()->value(),

rect.width(), rect.height());

else

return rect;

}

QRegion PieView::*visualRegionForSelection*(const QItemSelection &selection) const

{

int ranges = selection.count();

if (ranges == 0)

return QRect();

QRegion region;

for (int i = 0; i < ranges; ++i) {

QItemSelectionRange range = selection.at(i);

for (int row = range.top(); row <= range.bottom(); ++row) {

for (int col = range.left(); col <= range.right(); ++col) {

QModelIndex index = model()->*index*(row, col, rootIndex());

region += *visualRect*(index);

}

}

}

return region;

}

## **源文件widget.cpp**

#include "widget.h"

#include "ui\_widget.h"

#include <QtSql>

#include <QtGui>

#include "pieview.h"

Widget::Widget(QWidget \*parent) :

QWidget(parent),

ui(new Ui::Widget)

{

ui->setupUi(this);

setFixedSize(845, 500);

ui->stackedWidget->setCurrentIndex(0);

QSqlQueryModel \*typeModel = new QSqlQueryModel(this);

typeModel->setQuery("select name from type");

ui->sellTypeComboBox->setModel(typeModel);

QSplitter \*splitter = new QSplitter(ui->managePage);

splitter->resize(700, 360);

splitter->move(0, 50);

splitter->addWidget(ui->toolBox);

splitter->addWidget(ui->dailyList);

splitter->setStretchFactor(0, 1);

splitter->setStretchFactor(1, 1);

on\_sellCancelBtn\_clicked();

on\_goodsCancelBtn\_clicked();

on\_newCancelBtn\_clicked();

showDailyList();

ui->typeComboBox->setModel(typeModel);

ui->goodsTypeComboBox->setModel(typeModel);

ui->newTypeComboBox->setModel(typeModel);

createChartModelView();

}

Widget::~*Widget*()

{

delete ui;

}

// 出售商品的商品类型改变时

void Widget::on\_sellTypeComboBox\_currentIndexChanged(QString type)

{

if (type == "请选择类型") {

// 进行其他部件的状态设置

on\_sellCancelBtn\_clicked();

} else {

ui->sellBrandComboBox->setEnabled(true);

QSqlQueryModel \*sellBrandModel = new QSqlQueryModel(this);

sellBrandModel->setQuery(QString("select name from brand where type='%1'").arg(type));

ui->sellBrandComboBox->setModel(sellBrandModel);

ui->sellCancelBtn->setEnabled(true);

}

}

// 已有商品入库的商品类型改变时

void Widget::on\_goodsTypeComboBox\_currentIndexChanged(QString type)

{

if (type == "请选择类型") {

// 进行其他部件的状态设置

on\_goodsCancelBtn\_clicked();

} else {

ui->goodsBrandComboBox->setEnabled(true);

QSqlQueryModel \*goodBrandModel = new QSqlQueryModel(this);

goodBrandModel->setQuery(QString("select name from brand where type='%1'").arg(type));

ui->goodsBrandComboBox->setModel(goodBrandModel);

ui->goodsCancelBtn->setEnabled(true);

}

}

// 新商品入库类型改变时

void Widget::on\_newTypeComboBox\_currentIndexChanged(QString type)

{

if (type == "请选择类型") {

// 进行其他部件的状态设置

on\_newCancelBtn\_clicked();

} else {

ui->newBrandLineEdit->setEnabled(true);

ui->newBrandLineEdit->setFocus();

}

}

// 出售商品的品牌改变时

void Widget::on\_sellBrandComboBox\_currentIndexChanged(QString brand)

{

ui->sellNumSpinBox->setValue(0);

ui->sellNumSpinBox->setEnabled(false);

ui->sellSumLineEdit->clear();

ui->sellSumLineEdit->setEnabled(false);

ui->sellOkBtn->setEnabled(false);

QSqlQuery query;

query.exec(QString("select price from brand where name='%1' and type='%2'")

.arg(brand).arg(ui->sellTypeComboBox->currentText()));

query.next();

ui->sellPriceLineEdit->setEnabled(true);

ui->sellPriceLineEdit->setReadOnly(true);

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

query.exec(QString("select last from brand where name='%1' and type='%2'")

.arg(brand).arg(ui->sellTypeComboBox->currentText()));

query.next();

int num = query.value(0).toInt();

if (num == 0) {

QMessageBox::information(this, tr("提示"), tr("该商品已经售完！"), QMessageBox::Ok);

} else {

ui->sellNumSpinBox->setEnabled(true);

ui->sellNumSpinBox->setMaximum(num);

ui->sellLastNumLabel->setText(tr("剩余数量：%1").arg(num));

ui->sellLastNumLabel->*setVisible*(true);

}

}

// 已有商品入库的品牌改变时

void Widget::on\_goodsBrandComboBox\_currentIndexChanged(QString brand)

{

ui->goodsNumSpinBox->setValue(0);

ui->goodsNumSpinBox->setEnabled(true);

ui->goodsSumLineEdit->clear();

ui->goodsSumLineEdit->setEnabled(false);

ui->goodsOkBtn->setEnabled(false);

QSqlQuery query;

query.exec(QString("select price from brand where name='%1' and type='%2'")

.arg(brand).arg(ui->goodsTypeComboBox->currentText()));

query.next();

ui->goodsPriceLineEdit->setEnabled(true);

ui->goodsPriceLineEdit->setReadOnly(true);

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

}

// 新商品品牌改变时

void Widget::on\_newBrandLineEdit\_textChanged(QString str)

{

if (str == "") {

ui->newCancelBtn->setEnabled(false);

ui->newPriceSpinBox->setEnabled(false);

ui->newNumSpinBox->setEnabled(false);

ui->newSumLineEdit->setEnabled(false);

ui->newSumLineEdit->clear();

ui->newOkBtn->setEnabled(false);

} else {

ui->newCancelBtn->setEnabled(true);

ui->newPriceSpinBox->setEnabled(true);

ui->newNumSpinBox->setEnabled(true);

ui->newSumLineEdit->setEnabled(true);

qreal sum = ui->newPriceSpinBox->value() \* ui->newNumSpinBox->value();

ui->newSumLineEdit->setText(QString::number(sum));

ui->newOkBtn->setEnabled(true);

}

}

// 出售商品数量改变时

void Widget::on\_sellNumSpinBox\_valueChanged(int value)

{

if (value == 0) {

ui->sellSumLineEdit->clear();

ui->sellSumLineEdit->setEnabled(false);

ui->sellOkBtn->setEnabled(false);

} else {

ui->sellSumLineEdit->setEnabled(true);

ui->sellSumLineEdit->setReadOnly(true);

qreal sum = value \* ui->sellPriceLineEdit->text().toInt();

ui->sellSumLineEdit->setText(QString::number(sum));

ui->sellOkBtn->setEnabled(true);

}

}

// 已有商品入库数量改变时

void Widget::on\_goodsNumSpinBox\_valueChanged(int value)

{

if (value == 0) {

ui->goodsSumLineEdit->clear();

ui->goodsSumLineEdit->setEnabled(false);

ui->goodsOkBtn->setEnabled(false);

} else {

ui->goodsSumLineEdit->setEnabled(true);

ui->goodsSumLineEdit->setReadOnly(true);

qreal sum = value \* ui->goodsPriceLineEdit->text().toInt();

ui->goodsSumLineEdit->setText(QString::number(sum));

ui->goodsOkBtn->setEnabled(true);

}

}

// 新商品单价改变时

void Widget::on\_newPriceSpinBox\_valueChanged(int value)

{

qreal sum = value \* ui->newNumSpinBox->value();

ui->newSumLineEdit->setText(QString::number(sum));

ui->newOkBtn->setEnabled(true);

}

// 新商品数量

void Widget::on\_newNumSpinBox\_valueChanged(int value)

{

qreal sum = value \* ui->newPriceSpinBox->value();

ui->newSumLineEdit->setText(QString::number(sum));

ui->newOkBtn->setEnabled(true);

}

// 出售商品的取消按钮

void Widget::on\_sellCancelBtn\_clicked()

{

ui->sellTypeComboBox->setCurrentIndex(0);

ui->sellBrandComboBox->clear();

ui->sellBrandComboBox->setEnabled(false);

ui->sellPriceLineEdit->clear();

ui->sellPriceLineEdit->setEnabled(false);

ui->sellNumSpinBox->setValue(0);

ui->sellNumSpinBox->setEnabled(false);

ui->sellSumLineEdit->clear();

ui->sellSumLineEdit->setEnabled(false);

ui->sellOkBtn->setEnabled(false);

ui->sellCancelBtn->setEnabled(false);

ui->sellLastNumLabel->*setVisible*(false);

}

// 已有商品入库的取消按钮

void Widget::on\_goodsCancelBtn\_clicked()

{

ui->goodsTypeComboBox->setCurrentIndex(0);

ui->goodsBrandComboBox->clear();

ui->goodsBrandComboBox->setEnabled(false);

ui->goodsPriceLineEdit->clear();

ui->goodsPriceLineEdit->setEnabled(false);

ui->goodsNumSpinBox->setValue(0);

ui->goodsNumSpinBox->setEnabled(false);

ui->goodsSumLineEdit->clear();

ui->goodsSumLineEdit->setEnabled(false);

ui->goodsOkBtn->setEnabled(false);

ui->goodsCancelBtn->setEnabled(false);

}

// 新商品入库的取消按钮

void Widget::on\_newCancelBtn\_clicked()

{

ui->newTypeComboBox->setCurrentIndex(0);

ui->newBrandLineEdit->clear();

ui->newBrandLineEdit->setEnabled(false);

ui->newPriceSpinBox->setEnabled(false);

ui->newPriceSpinBox->setValue(1);

ui->newNumSpinBox->setEnabled(false);

ui->newNumSpinBox->setValue(1);

ui->newSumLineEdit->setText("1");

ui->newSumLineEdit->setEnabled(false);

ui->newOkBtn->setEnabled(false);

ui->newCancelBtn->setEnabled(false);

}

// 出售商品的确定按钮

void Widget::on\_sellOkBtn\_clicked()

{

QString type = ui->sellTypeComboBox->currentText();

QString name = ui->sellBrandComboBox->currentText();

int value = ui->sellNumSpinBox->value();

// cellNumSpinBox的最大值就是以前的剩余量

int last = ui->sellNumSpinBox->maximum() - value;

QSqlQuery query;

// 获取以前的销售量

query.exec(QString("select sell from brand where name='%1' and type='%2'")

.arg(name).arg(type));

query.next();

int sell = query.value(0).toInt() + value;

// 事务操作

QSqlDatabase::database().transaction();

bool rtn = query.exec(

QString("update brand set sell=%1,last=%2 where name='%3' and type='%4'")

.arg(sell).arg(last).arg(name).arg(type));

if (rtn) {

QSqlDatabase::database().commit();

QMessageBox::information(this, tr("提示"), tr("购买成功！"), QMessageBox::Ok);

writeXml();

showDailyList();

} else {

QSqlDatabase::database().rollback();

QMessageBox::information(this, tr("提示"), tr("购买失败，无法访问数据库！"), QMessageBox::Ok);

}

on\_sellCancelBtn\_clicked();

}

// 已有商品入库的确定按钮

void Widget::on\_goodsOkBtn\_clicked()

{

QString type = ui->goodsTypeComboBox->currentText();

QString name = ui->goodsBrandComboBox->currentText();

int value = ui->goodsNumSpinBox->value();

QSqlQuery query;

// 获取以前的总量

query.exec(QString("select sum from brand where name='%1' and type='%2'")

.arg(name).arg(type));

query.next();

int sum = query.value(0).toInt() + value;

// 获取以前的剩余量

query.exec(QString("select last from brand where name='%1' and type='%2'")

.arg(name).arg(type));

query.next();

int last = query.value(0).toInt() + value;

// 事务操作

QSqlDatabase::database().transaction();

bool rtn = query.exec(

QString("update brand set sum=%1,last=%2 where name='%3' and type='%4'")

.arg(sum).arg(last).arg(name).arg(type));

if (rtn) {

QSqlDatabase::database().commit();

QMessageBox::information(this, tr("提示"), tr("入库成功！"), QMessageBox::Ok);

} else {

QSqlDatabase::database().rollback();

QMessageBox::information(this, tr("提示"), tr("入库失败，无法访问数据库！"), QMessageBox::Ok);

}

on\_goodsCancelBtn\_clicked();

}

// 新商品的确定按钮

void Widget::on\_newOkBtn\_clicked()

{

QString type = ui->newTypeComboBox->currentText();

QString brand = ui->newBrandLineEdit->text();

qint16 price = ui->newPriceSpinBox->value();

qint16 num = ui->newNumSpinBox->value();

QSqlQuery query;

query.exec("select id from brand");

query.last();

qreal temp = query.value(0).toInt() + 1;

QString id;

if (temp < 10) {

id = "0" + QString::number(temp);

} else {

id = QString::number(temp);

}

qDebug() << "hello" <<id << type << brand << price << num;

// 事务操作

QSqlDatabase::database().transaction();

bool rtn = query.exec(QString("insert into brand values('%1', '%2', '%3', %4, %5, 0, %6)")

.arg(id).arg(brand).arg(type).arg(price).arg(num).arg(num));

if (rtn) {

QSqlDatabase::database().commit();

QMessageBox::information(this, tr("提示"), tr("入库成功！"), QMessageBox::Ok);

} else {

QSqlDatabase::database().rollback();

QMessageBox::information(this, tr("提示"), tr("入库失败，无法访问数据库！"), QMessageBox::Ok);

}

on\_newCancelBtn\_clicked();

}

// 获取当前的日期或者时间

QString Widget::getDateTime(Widget::DateTimeType type)

{

QDateTime datetime = QDateTime::currentDateTime();

QString date = datetime.toString("yyyy-MM-dd");

QString time = datetime.toString("hh:mm");

QString dateAndTime = datetime.toString("yyyy-MM-dd dddd hh:mm");

if(type == Date) return date;

else if(type == Time) return time;

else return dateAndTime;

}

// 读取XML文档

bool Widget::docRead()

{

QFile file("data.xml");

if(!file.*open*(QIODevice::ReadOnly))

return false;

if(!doc.setContent(&file)){

file.*close*();

return false;

}

file.*close*();

return true;

}

// 写入xml文档

bool Widget::docWrite()

{

QFile file("data.xml");

if(!file.*open*(QIODevice::WriteOnly | QIODevice::Truncate))

return false;

QTextStream out(&file);

doc.save(out,4);

file.*close*();

return true;

}

// 将销售记录写入文档

void Widget::writeXml()

{

// 先从文件读取

if (docRead()) {

QString currentDate = getDateTime(Date);

QDomElement root = doc.documentElement();

// 根据是否有日期节点进行处理

if (!root.hasChildNodes()) {

QDomElement date = doc.createElement(QString("日期"));

QDomAttr curDate = doc.createAttribute("date");

curDate.setValue(currentDate);

date.setAttributeNode(curDate);

root.appendChild(date);

createNodes(date);

} else {

QDomElement date = root.lastChild().toElement();

// 根据是否已经有今天的日期节点进行处理

if (date.attribute("date") == currentDate) {

createNodes(date);

} else {

QDomElement date = doc.createElement(QString("日期"));

QDomAttr curDate = doc.createAttribute("date");

curDate.setValue(currentDate);

date.setAttributeNode(curDate);

root.appendChild(date);

createNodes(date);

}

}

// 写入到文件

docWrite();

}

}

// 创建节点

void Widget::createNodes(QDomElement &date)

{

QDomElement time = doc.createElement(QString("时间"));

QDomAttr curTime = doc.createAttribute("time");

curTime.setValue(getDateTime(Time));

time.setAttributeNode(curTime);

date.appendChild(time);

QDomElement type = doc.createElement(QString("类型"));

QDomElement brand = doc.createElement(QString("品牌"));

QDomElement price = doc.createElement(QString("单价"));

QDomElement num = doc.createElement(QString("数量"));

QDomElement sum = doc.createElement(QString("金额"));

QDomText text;

text = doc.createTextNode(QString("%1")

.arg(ui->sellTypeComboBox->currentText()));

type.appendChild(text);

text = doc.createTextNode(QString("%1")

.arg(ui->sellBrandComboBox->currentText()));

brand.appendChild(text);

text = doc.createTextNode(QString("%1")

.arg(ui->sellPriceLineEdit->text()));

price.appendChild(text);

text = doc.createTextNode(QString("%1")

.arg(ui->sellNumSpinBox->value()));

num.appendChild(text);

text = doc.createTextNode(QString("%1")

.arg(ui->sellSumLineEdit->text()));

sum.appendChild(text);

time.appendChild(type);

time.appendChild(brand);

time.appendChild(price);

time.appendChild(num);

time.appendChild(sum);

}

// 显示日销售清单

void Widget::showDailyList()

{

ui->dailyList->clear();

if (docRead()) {

QDomElement root = doc.documentElement();

QString title = root.tagName();

QListWidgetItem \*titleItem = new QListWidgetItem;

titleItem->setText(QString("-----%1-----").arg(title));

titleItem->setTextAlignment(Qt::AlignCenter);

ui->dailyList->addItem(titleItem);

if (root.hasChildNodes()) {

QString currentDate = getDateTime(Date);

QDomElement dateElement = root.lastChild().toElement();

QString date = dateElement.attribute("date");

if (date == currentDate) {

ui->dailyList->addItem("");

ui->dailyList->addItem(QString("日期：%1").arg(date));

ui->dailyList->addItem("");

QDomNodeList children = dateElement.childNodes();

// 遍历当日销售的所有商品

for (int i=0; i<children.count(); i++) {

QDomNode node = children.at(i);

QString time = node.toElement().attribute("time");

QDomNodeList list = node.childNodes();

QString type = list.at(0).toElement().text();

QString brand = list.at(1).toElement().text();

QString price = list.at(2).toElement().text();

QString num = list.at(3).toElement().text();

QString sum = list.at(4).toElement().text();

QString str = time + " 出售 " + brand + type

+ " " + num + "台， " + "单价：" + price

+ "元， 共" + sum + "元";

QListWidgetItem \*tempItem = new QListWidgetItem;

tempItem->setText("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

tempItem->setTextAlignment(Qt::AlignCenter);

ui->dailyList->addItem(tempItem);

ui->dailyList->addItem(str);

}

}

}

}

}

// 创建销售记录图表的模型和视图

void Widget::createChartModelView()

{

chartModel = new QStandardItemModel(this);

chartModel->setColumnCount(2);

chartModel->*setHeaderData*(0, Qt::Horizontal, QString("品牌"));

chartModel->*setHeaderData*(1, Qt::Horizontal, QString("销售数量"));

QSplitter \*splitter = new QSplitter(ui->chartPage);

splitter->resize(700, 320);

splitter->move(0, 80);

QTableView \*table = new QTableView;

PieView \*pieChart = new PieView;

splitter->addWidget(table);

splitter->addWidget(pieChart);

splitter->setStretchFactor(0, 1);

splitter->setStretchFactor(1, 2);

table->*setModel*(chartModel);

pieChart->*setModel*(chartModel);

QItemSelectionModel \*selectionModel = new QItemSelectionModel(chartModel);

table->*setSelectionModel*(selectionModel);

pieChart->*setSelectionModel*(selectionModel);

}

// 显示销售记录图表

void Widget::showChart()

{

QSqlQuery query;

query.exec(QString("select name,sell from brand where type='%1'")

.arg(ui->typeComboBox->currentText()));

chartModel->*removeRows*(0, chartModel->*rowCount*(QModelIndex()), QModelIndex());

int row = 0;

while(query.next()) {

int r = qrand() % 256;

int g = qrand() % 256;

int b = qrand() % 256;

chartModel->*insertRows*(row, 1, QModelIndex());

chartModel->*setData*(chartModel->*index*(row, 0, QModelIndex()),

query.value(0).toString());

chartModel->*setData*(chartModel->*index*(row, 1, QModelIndex()),

query.value(1).toInt());

chartModel->*setData*(chartModel->*index*(row, 0, QModelIndex()),

QColor(r, g, b), Qt::DecorationRole);

row++;

}

}

// 销售统计页面的类型选择框

void Widget::on\_typeComboBox\_currentIndexChanged(QString type)

{

if (type != "请选择类型")

showChart();

}

// 更新显示按钮

void Widget::on\_updateBtn\_clicked()

{

if (ui->typeComboBox->currentText() != "请选择类型")

showChart();

}

// 商品管理按钮

void Widget::on\_manageBtn\_clicked()

{

ui->stackedWidget->setCurrentIndex(0);

}

// 销售统计按钮

void Widget::on\_chartBtn\_clicked()

{

ui->stackedWidget->setCurrentIndex(1);

}

// 修改密码按钮

void Widget::on\_passwordBtn\_clicked()

{

ui->stackedWidget->setCurrentIndex(2);

}

// 修改密码确定按钮

void Widget::on\_changePwdBtn\_clicked()

{

if (ui->oldPwdLineEdit->text().isEmpty() ||

ui->newPwdLineEdit->text().isEmpty() ||

ui->newPwdLineEdit\_2->text().isEmpty()) {

QMessageBox::warning(this, tr("警告"), tr("请将信息填写完整！"),

QMessageBox::Ok);

} else {

if (ui->newPwdLineEdit->text() == ui->newPwdLineEdit\_2->text()) {

QSqlQuery query;

query.exec("select pwd from password");

query.next();

if (query.value(0).toString() == ui->oldPwdLineEdit->text()) {

bool temp = query.exec(QString("update password set pwd='%1' where pwd='%2'")

.arg(ui->newPwdLineEdit->text()).arg(ui->oldPwdLineEdit->text()));

if (temp) {

QMessageBox::information(this, tr("提示"), tr("密码修改成功！"),

QMessageBox::Ok);

ui->oldPwdLineEdit->clear();

ui->newPwdLineEdit->clear();

ui->newPwdLineEdit\_2->clear();

} else {

QMessageBox::information(this, tr("提示"), tr("密码修改失败，无法访问数据库！"),

QMessageBox::Ok);

}

} else {

QMessageBox::warning(this, tr("警告"), tr("原密码错误，请重新填写！"),

QMessageBox::Ok);

ui->oldPwdLineEdit->clear();

ui->newPwdLineEdit->clear();

ui->newPwdLineEdit\_2->clear();

ui->oldPwdLineEdit->setFocus();

}

} else {

QMessageBox::warning(this, tr("警告"), tr("两次密码输入不一致，请重新填写！"),QMessageBox::Ok);

ui->oldPwdLineEdit->clear();

ui->newPwdLineEdit->clear();

ui->newPwdLineEdit\_2->clear();

ui->oldPwdLineEdit->setFocus();

}

}

}

//关于按钮

void Widget::on\_aboutBtn\_clicked()

{

ui->stackedWidget->setCurrentIndex(3);

}

**资源文件 略**

**界面文件 略**