Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

о лабораторной работе

Тема: «Автоматизированное рабочее место менеджера по продажам»

Выполнил

Студент группы РИС-23-1б

Валинуров Р.Н.

Проверила доц. кафедры ИТАС

Полякова О.А.

Пермь, 2024

**Содержание**

Введение**3**

1 Постановка задачи**4**

2 Анализ задачи**5-6**

3 Разработка программы**7**

Заключение**8**

Использованные источники**9**

Приложение А**10-38**

**Введение**

В современном мире информационные технологии играют ключевую роль в оптимизации бизнес-процессов и повышении эффективности работы различных подразделений компаний. Одной из важных задач, стоящих перед организациями, является автоматизация рабочих мест сотрудников, что позволяет сократить затраты времени на выполнение рутинных операций и повысить точность выполняемых задач. В частности, автоматизация рабочего места менеджера по продажам является приоритетной задачей для многих компаний, стремящихся улучшить процессы управления клиентами и увеличения объемов продаж.

Цель данной лабораторной работы заключается в разработке и внедрении автоматизированного рабочего места менеджера по продажам с использованием базы данных SQLite. SQLite выбрана в качестве СУБД (системы управления базами данных) благодаря своей легкости, высокой производительности и простоте интеграции в различные программные продукты. В рамках лабораторной работы будет рассмотрен процесс создания базы данных, проектирование её структуры, а также разработка программного обеспечения для взаимодействия с этой базой данных.

**1 Постановка задачи**

Целью данной лабораторной работы является создание автоматизированного рабочего места менеджера по продажам с использованием базы данных SQLite. Для этого необходимо:

Анализ требований:

* + Определить ключевые функции менеджера по продажам, требующие автоматизации.

Проектирование базы данных:

* + Разработать структуру базы данных, включающую таблицы для продаж, товаров и услуг.
  + Определить связи между таблицами.

Разработка ПО:

* + Создать программное обеспечение для работы с базой данных SQLite.
  + Реализовать функции добавления, изменения, удаления и поиска данных.
  + Разработать пользовательский интерфейс для менеджера по продажам.

**2 Анализ задачи**

Для разработки автоматизированного рабочего места менеджера по продажам с использованием SQLite необходимо выполнить следующие шаги:

Основные функции менеджера по продажам

* Ведение учета продаж (обновление статуса).
* Управление товарами и услугами (ведение каталога, обновление данных).
* Анализ и отчетность (формирование отчетов, анализ данных).

Ключевые процессы для автоматизации

Управление продажами:

* + Регистрация новых сделок.
  + Обновление статуса существующих продаж.
  + Хранение информации о проведенных сделках и их результатах.

Управление товарами и услугами:

* + Ведение каталога товаров и услуг.
  + Обновление информации о продуктах (цены, характеристики, наличие).

Анализ и отчетность:

* + Формирование отчетов по продажам за определенные периоды.
  + Анализ данных для выявления тенденций и принятия стратегических решений.

Требования к функциональности системы

База данных:

* + Использование SQLite для хранения информации о продажах и продуктах.
  + Оптимизация структуры базы данных для обеспечения быстрого доступа и обработки данных.

Интерфейс пользователя:

* + Удобный и интуитивно понятный интерфейс для менеджера по продажам.
  + Возможность быстрого доступа к основным функциям системы.

Функциональность ПО:

* + операции (создание, чтение, обновление, удаление) для работы с данными о продажах и товарах.
  + Поиск и фильтрация данных.
  + Формирование отчетов.

**3 Разработка программы**

Разработка програмы будет происходить на языке программировании С++ в среде разработки Qt Creator 5.0.2 (Community)

Для начала разработки программы, которая будет автоматизировать рабочее место специалиста, необходимо написать класс, который будут отвечать за формирование самой базы данных, таким классом будет mydb. В классе описана функция init(), которая отвечает за путь к файлу базы данных SQLite, деструктор класса будет закрывать базу данных, при этом сохраняя в файле базы данных все введенные значения. Далее необходимо написать класс, который будет отвечать за ввод данных в базу. В классе additem описан конструктор, вызывающий диалоговое окно, в котором необходимо ввести все данные, после ввода нажимается кнопка “Добавить”. Класс finditem отвечает за поиск продукта в базе данных по одному из введенных данных: id, наименование, статус, цена продажи. Класс itemsreport отвечает за вызов отчета, т.е. самой базы данных. Также в программу были добавлены классы, отвечающие за формирование базы данных по определенному статусу, так в программе реализованы классы availableitems, expireditems, returneditems, solditems, spoileditems. Для изменения статуса продукта, например если он сначала был на складе, а потом был продан, формируется класс updateitem, в котором описана функция, отвечающая за смену статуса продукта. В классе mainwindow подключены заголовочные файлы всех классов, при начале программы формируется окно со всеми заданными функциями.

**Заключение**

В данной лабораторной работе было разработано автоматизированное рабочее место менеджера по продажам с использованием базы данных SQLite. Разработанная система позволяет менеджерам по продажам эффективно управлять сделками и каталогом товаров, а также формировать отчеты для анализа. Это способствует повышению производительности и точности работы, улучшая бизнес-процессы компании.

**Список использованных источников**

1. Технологии разработки объектно-ориентированных программ на языке С++ : учеб. пособие : в 3 ч. / О.А. Полякова, О.Л. Викентьева.
2. Жасмин Бланшет, Марк Саммерфилд Qt 4: программирование GUI на C++
3. Документация Qt Creator [Электронный ресурс]: <https://doc.qt.io/qtcreator/>

**ПРИЛОЖЕНИЕ А**

**Исходный код программы**

#include "additem.h"

#include "ui\_additem.h"

AddItem::**AddItem**(QWidget \*parent) :

QDialog(*parent*),

ui(*new* Ui::AddItem)

{

ui->setupUi(*this*);

}

AddItem::~***AddItem***()

{

*delete* ui;

}

void AddItem::**Show**()

{

*this*->show();

}

void AddItem::**on\_btnAdd\_clicked**()

{

ui->lblInfo->setText("");

QString sId = ui->txtId->text();

QString sName = ui->txtName->text();

QString sQuantity = ui->txtQuantity->text();

QString sStatus = ui->cmbStatus->currentText();

QString sDOP = ui->txtDOP->text();

QString sDOS = ui->txtDOS->text();

QString sSellingPrice = ui->txtSellingPrice->text();

QString sPurchasePrice = ui->txtPurchasePrice->text();

QSqlQuery query(MyDB::getInstance()->getDBInstance());

query.clear();

query.prepare("insert into database(Id, Name, Quantity, Status, DOP, DOS, SellingPrice, PurchasePrice) values ('"+sId +"','" +sName +"','" +sQuantity + "','"+ sStatus + "','" +sDOP + "','"+ sDOS+ "','"+sSellingPrice+ "','"+ sPurchasePrice + "')");

*if*(!query.exec())

{

qDebug() << query.lastError().text() << query.lastQuery();

ui->lblInfo->setText("Невозможно добавить");

}

*else*

{

*if*(query.numRowsAffected() > 0)

{

qDebug() <<"read was successful "<< query.lastQuery();

ui->lblInfo->setText("Продукт добавлен!");

resetElements();

}

*else*

qDebug() <<"Unable to Add new Item";

}

}

void AddItem::**resetElements**()

{

ui->txtId->clear();

ui->txtName->clear();

ui->txtQuantity->clear();

ui->txtDOP->clear();

ui->txtDOS->clear();

ui->txtSellingPrice->clear();

ui->txtPurchasePrice->clear();

}

#include "availableitems.h"

#include "ui\_availableitems.h"

AvailableItems::AvailableItems(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::AvailableItems)

{

ui->setupUi(*this*);

}

AvailableItems::~AvailableItems()

{

*delete* ui;

}

void AvailableItems::Show()

{

updateUI();

show();

}

void AvailableItems::updateUI()

{

qDebug() << "in init()";

QSqlQueryModel \* model = *new* QSqlQueryModel(*this*);

QSqlQuery query( MyDB::getInstance()->getDBInstance());

query.prepare("select \* from database where Status='На складе'");

*if*(!query.exec())

qDebug() << query.lastError().text() << query.lastQuery();

*else*

qDebug() << "== success query fetch()";

*while*(query.next())

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

model->setQuery(query);

ui->tableView->setModel(model);

qDebug() << "rows are : " << model->rowCount();

ui->tableView->show();

}

#include "expireditems.h"

#include "ui\_expireditems.h"

ExpiredItems::ExpiredItems(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::ExpiredItems)

{

ui->setupUi(*this*);

}

ExpiredItems::~ExpiredItems()

{

*delete* ui;

}

void ExpiredItems::Show()

{

updateUI();

show();

}

void ExpiredItems::updateUI()

{

qDebug() << "in init()";

QSqlQueryModel \* model = *new* QSqlQueryModel(*this*);

QSqlQuery query( MyDB::getInstance()->getDBInstance());

query.prepare("select \* from database where Status='Истек срок годности'");

*if*(!query.exec())

qDebug() << query.lastError().text() << query.lastQuery();

*else*

qDebug() << "== success query fetch()";

*while*(query.next())

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

model->setQuery(query);

ui->tableView->setModel(model);

qDebug() << "rows are : " << model->rowCount();

ui->tableView->show();

}

#include "finditem.h"

#include "ui\_finditem.h"

FindItem::**FindItem**(QWidget \*parent) :

QDialog(*parent*),

ui(*new* Ui::FindItem)

{

ui->setupUi(*this*);

ui->rdoName->setChecked(*true*);

}

FindItem::~***FindItem***()

{

*delete* ui;

}

void FindItem::**on\_btnFind\_clicked**()

{

QString sValue = ui->txtValue->text();

QString sCriteriaColum = "";

*if*(ui->rdoId->isChecked())

sCriteriaColum="Id";

*else* *if* (ui->rdoName->isChecked())

sCriteriaColum="Name";

*else* *if* (ui->rdoQuantity->isChecked())

sCriteriaColum="Quantity";

*else* *if* (ui->rdoSellingPrice->isChecked())

sCriteriaColum = "SellingPrice";

*else*

sCriteriaColum = "Name";

qDebug() << "in init()";

QSqlQueryModel \* model = *new* QSqlQueryModel(*this*);

QSqlQuery query( MyDB::getInstance()->getDBInstance());

query.prepare("select \* from database where " +sCriteriaColum+" like '%" + sValue + "%'");

*if*(!query.exec())

qDebug() << query.lastError().text() << query.lastQuery();

*else*

{

*if*(query.numRowsAffected()>0)

qDebug() << "== success query fetch() : " << query.lastQuery();

*else* {

qDebug() << " No record ! found ";

qDebug() << " Query is : " << query.lastQuery();

}

}

*while*(query.next())

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

model->setQuery(query);

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

qDebug() << "rows are : " << model->*rowCount*();

ui->tableView->show();

}

#include "itemsreport.h"

#include "ui\_itemsreport.h"

ItemsReport::**ItemsReport**(QWidget \*parent) :

QDialog(*parent*),

ui(*new* Ui::ItemsReport)

{

ui->setupUi(*this*);

}

ItemsReport::~***ItemsReport***()

{

*delete* ui;

}

void ItemsReport::**Show**()

{

updateUI();

*this*->show();

}

void ItemsReport::**updateUI**()

{

qDebug() << "in init()";

QSqlQueryModel \* model = *new* QSqlQueryModel(*this*);

QSqlQuery query( MyDB::getInstance()->getDBInstance());

query.prepare("select \* from database order by Id desc");

*if*(!query.exec())

qDebug() << query.lastError().text() << query.lastQuery();

*else*

qDebug() << "== success query fetch()";

*while*(query.next())

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

model->setQuery(query);

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

qDebug() << "rows are : " << model->*rowCount*();

ui->tableView->show();

}

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, *argv*);

MainWindow w;

w.show();

*return* a.exec();

}

#include "mainwindow.h"

#include "ui\_mainwindow.h"

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

QMainWindow(*parent*),

ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

ptrAddItem = *new* AddItem(*this*);

ptrItemReport = *new* ItemsReport(*this*);

ptrSpoiledItems = *new* SpoiledItems(*this*);

ptrSoldITems = *new* SoldItems(*this*);

ptrExpiredItems = *new* ExpiredItems(*this*);

ptrReturnedItems = *new* ReturnedItems(*this*);

ptrAvailableItems = *new* AvailableItems(*this*);

ptrUpdateItem = *new* UpdateItem(*this*);

ptrFindItem = *new* FindItem(*this*);

*this*->setFixedSize(640,260);

}

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

{

*delete* ui;

*delete* ptrAddItem;

*delete* ptrItemReport;

*delete* ptrSpoiledItems;

*delete* ptrSoldITems;

*delete* ptrExpiredItems;

*delete* ptrReturnedItems;

*delete* ptrFindItem;

*delete* ptrUpdateItem;

MyDB::ResetInstance();

}

void MainWindow::**on\_btnAddItem\_clicked**()

{

ptrAddItem->Show();

}

void MainWindow::**on\_btnUpdateItem\_clicked**()

{

ptrUpdateItem->Show();

}

void MainWindow::**on\_btnSoldItems\_clicked**()

{

ptrSoldITems->Show();

}

void MainWindow::**on\_btnReport\_clicked**()

{

ptrItemReport->Show();

}

void MainWindow::**on\_btnExpiredItems\_clicked**()

{

ptrExpiredItems->Show();

}

void MainWindow::**on\_btnAvailableItems\_clicked**()

{

ptrAvailableItems->Show();

}

void MainWindow::**on\_btnSpoiledItem\_clicked**()

{

ptrSpoiledItems->Show();

}

void MainWindow::**on\_btnReturnItems\_clicked**()

{

ptrReturnedItems->Show();

}

void MainWindow::**on\_btnFindItem\_clicked**()

{

ptrFindItem->show();

}

#include "mydb.h"

MyDB\* MyDB::instance = *nullptr*;

MyDB::**MyDB**()

{

init();

}

void MyDB::**init**()

{

qDebug() << "in init()";

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

db.setDatabaseName("c:/Users/hbyfn/Desktop/Grocery\_Store\_Management/manager\_database.sqlite");

*if*(QFile::exists("c:/Users/hbyfn/Desktop/Grocery\_Store\_Management/manager\_database.sqlite"))

qDebug() << "DB file exist";

*else*

qDebug() << "DB file doesn't exists";

*if* (!db.open())

qDebug() << db.lastError().text();

*else*

qDebug() << "Database loaded successfull!";

}

MyDB \*MyDB::**getInstance**()

{

qDebug() << "in MyDB::getInstance()";

*if*(instance == *nullptr*)

instance = *new* MyDB();

*return* instance;

}

QSqlDatabase MyDB::**getDBInstance**()

{

qDebug() <<"in getDBInstance()";

*return* db;

}

void MyDB::**ResetInstance**()

{

qDebug() << "deleting instance of MyDB";

*delete* instance;

instance = *nullptr*;

}

MyDB::~**MyDB**()

{

qDebug() << "closing database connection";

db.close();

}

#include "returneditems.h"

#include "ui\_returneditems.h"

ReturnedItems::ReturnedItems(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::ReturnedItems)

{

ui->setupUi(*this*);

}

ReturnedItems::~ReturnedItems()

{

*delete* ui;

}

void ReturnedItems::Show()

{

updateUI();

show();

}

void ReturnedItems::updateUI()

{

qDebug() << "in init()";

QSqlQueryModel \* model = *new* QSqlQueryModel(*this*);

QSqlQuery query( MyDB::getInstance()->getDBInstance());

query.prepare("select \* from database where Status='Возвращен'");

*if*(!query.exec())

qDebug() << query.lastError().text() << query.lastQuery();

*else*

qDebug() << "== success query fetch()";

*while*(query.next())

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

model->setQuery(query);

ui->tableView->setModel(model);

qDebug() << "rows are : " << model->rowCount();

ui->tableView->show();

}

#include "solditems.h"

#include "ui\_solditems.h"

SoldItems::SoldItems(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::SoldItems)

{

ui->setupUi(*this*);

}

SoldItems::~SoldItems()

{

*delete* ui;

}

void SoldItems::Show()

{

updateUI();

*this*->show();

}

void SoldItems::updateUI()

{

qDebug() << "in init()";

QSqlQueryModel \* model = *new* QSqlQueryModel(*this*);

QSqlQuery query( MyDB::getInstance()->getDBInstance());

query.prepare("select \* from database where Status='Продан'");

*if*(!query.exec())

qDebug() << query.lastError().text() << query.lastQuery();

*else*

qDebug() << "== success query fetch()";

*while*(query.next())

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

model->setQuery(query);

ui->tableView->setModel(model);

qDebug() << "rows are : " << model->rowCount();

ui->tableView->show();

}

#include "spoileditems.h"

#include "ui\_spoileditems.h"

SpoiledItems::SpoiledItems(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::SpoiledItems)

{

ui->setupUi(*this*);

}

SpoiledItems::~SpoiledItems()

{

*delete* ui;

}

void SpoiledItems::Show()

{

updateUI();

*this*->show();

}

void SpoiledItems::updateUI()

{

qDebug() << "in init()";

QSqlQueryModel \* model = *new* QSqlQueryModel(*this*);

QSqlQuery query( MyDB::getInstance()->getDBInstance());

query.prepare("select \* from database where Status='Брак'");

*if*(!query.exec())

qDebug() << query.lastError().text() << query.lastQuery();

*else*

qDebug() << "== success query fetch()";

*while*(query.next())

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

model->setQuery(query);

ui->tableView->setModel(model);

qDebug() << "rows are : " << model->rowCount();

ui->tableView->show();

}

#include "updateitem.h"

#include "ui\_updateitem.h"

UpdateItem::**UpdateItem**(QWidget \*parent) :

QDialog(*parent*),

ui(*new* Ui::UpdateItem)

{

ui->setupUi(*this*);

}

UpdateItem::~***UpdateItem***()

{

*delete* ui;

}

void UpdateItem::**updateUI**()

{

}

void UpdateItem::**Show**()

{

updateUI();

*this*->show();

}

void UpdateItem::**on\_pushButton\_clicked**()

{

ui->lblInfo->clear();

QString sId = ui->txtId->text();

QString sStatus = ui->cmbStatus->currentText();

QSqlQuery query( MyDB::getInstance()->getDBInstance());

query.prepare("update database set Status='" + sStatus + "' where Id = '" + sId + "'");

*if*(!query.exec())

{

qDebug() << query.lastError().text() << query.lastQuery();

ui->lblInfo->setText("Unable to update Item" + query.lastError().text());

}

*else*{

*if*(query.numRowsAffected()>0)

{

qDebug() << "Item Updated Successfully!";

ui->lblInfo->setText("Обновление продуктов успешно!");

}

*else*

{

qDebug() << " Unable to update Item";

ui->lblInfo->setText("Невозможно обновить!");

}

}

}

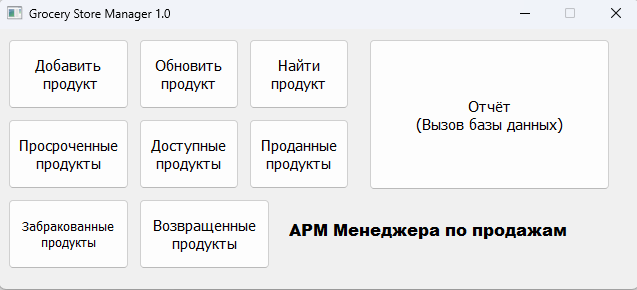
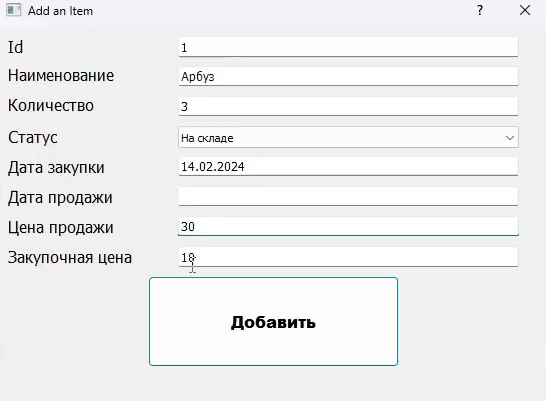
****

Рисунок 1 – Начальное окно программы



Рисунк 2 – Добавление продукта в базу данных

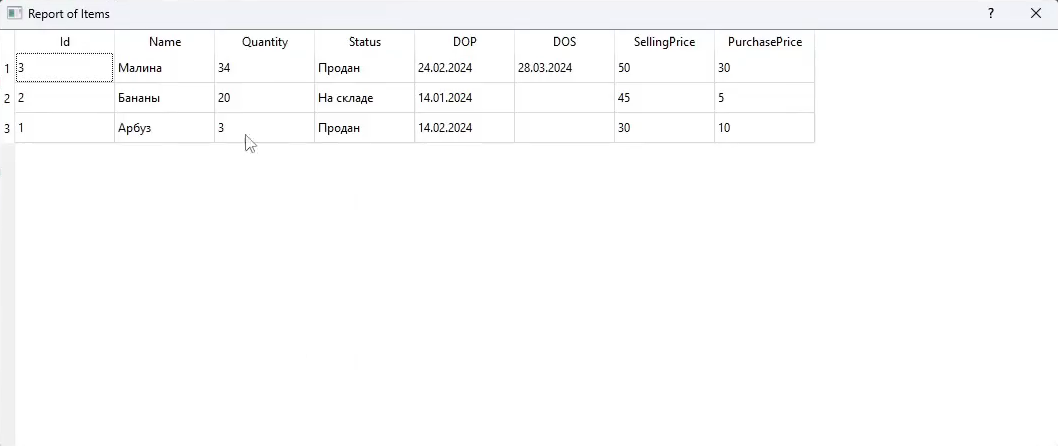
****

Рисунок 3 – Вывод базы данных

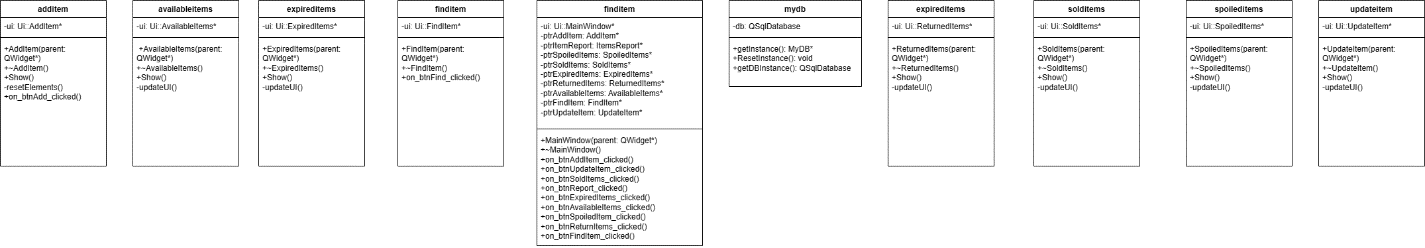


Рисунок 4 – UML диаграмма программы