



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт кибербезопасности и цифровых технологий

Кафедра КБ-9 «Предметно-ориентированные информационные системы»

Лабораторная работа

по дисциплине

«Управление данными»

наименование дисциплины

Тема лабораторной работы «Разработка приложения для работы с базой данных»

Студент группы БСБО-07-22
(учебная группа)

Борлыкова З. С.
Фамилия И.О.

Москва, 2023г.

Лабораторная работа №1. Подключение разрабатываемого приложения к БД

В ходе лабораторной работы был создан проект QtCourse, в котором будет разрабатываться приложение для работы с СУБД.

1. Было создано диалоговое окно для ввода строки подключения (Рис.1.1):

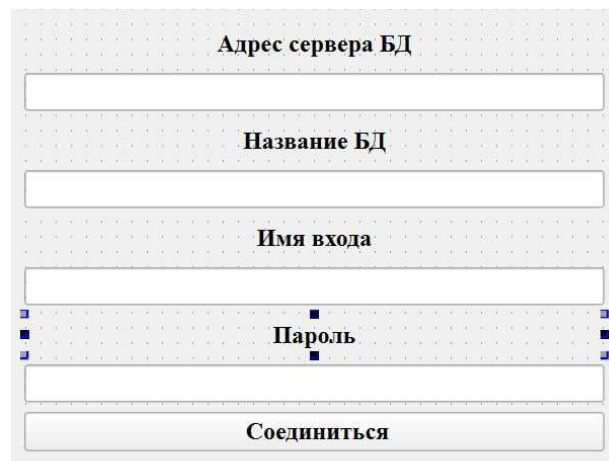


Рис.1.1 – Диалоговое окно login.ui

2. Написан метод, осуществляющий подключение к произвольной БД:

В файл mainwindow.h были подключены следующие библиотеки:

- QSqlDatabase - класс из библиотеки QtSql, содержащий необходимые для установки соединения методы;
- QMessageBox - класс, содержащий все необходимые методы для работы с простыми текстовыми сообщениями во всплывающих диалоговых окнах.

Для перехода к диалоговому окну подключения, на окно mainwindow.ui в меню была добавлена кнопка «Соединение с БД» (Рис.1.2):

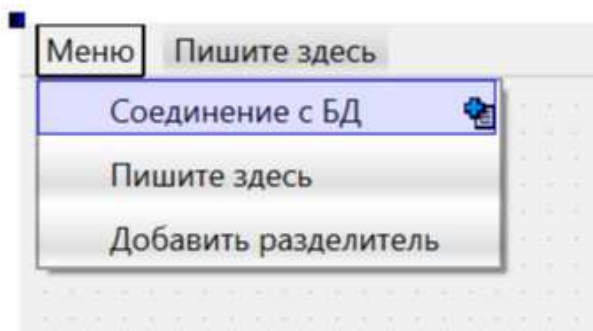


Рис. 1.2 – Изменение главного окна

Далее представлен листинг описания обработчика нажатия на кнопку «Подключиться» окна login.ui (Листинг 1.1).

Листинг 1.1- Метод, осуществляющий подключение к БД

```
void login::on_pushButton_clicked()
{
    db = QSqlDatabase::addDatabase("QODBC");
    db.setDatabaseName("DRIVER={SQL Server};
SERVER="+ui->lineEdit->text()+" ;DATABASE="+ui->lineEdit_2->text()+"");
    db.setUserName(ui->lineEdit_3->text());
    db.setPassword(ui->lineEdit_4->text());

    if(db.open())
    {
        mes->setText("Соединилось");
    }
    else
    {
        mes->setText("Соединение НЕ установлено");
    }

    mes->show();
}
```

3. В файле mainwindow.h были подключены ранее созданные модули, а также указатель на объект класса “login”. Чтобы сделать обработчик кнопки для перехода к окну подключения, был описан метод void MainWindow::on_action_triggered() в файле mainwindow.cpp(Листинг 1.2):

Листинг 1.2 – Описание обработчика метода нажатия на кнопку «Подключиться к БД»

```
void MainWindow::on_action_triggered()
{
    logwin = new login();
    logwin->show();
}
```

По итогу выполнения лабораторной работы было выполнено тестовое подключение (Рис.1.3):

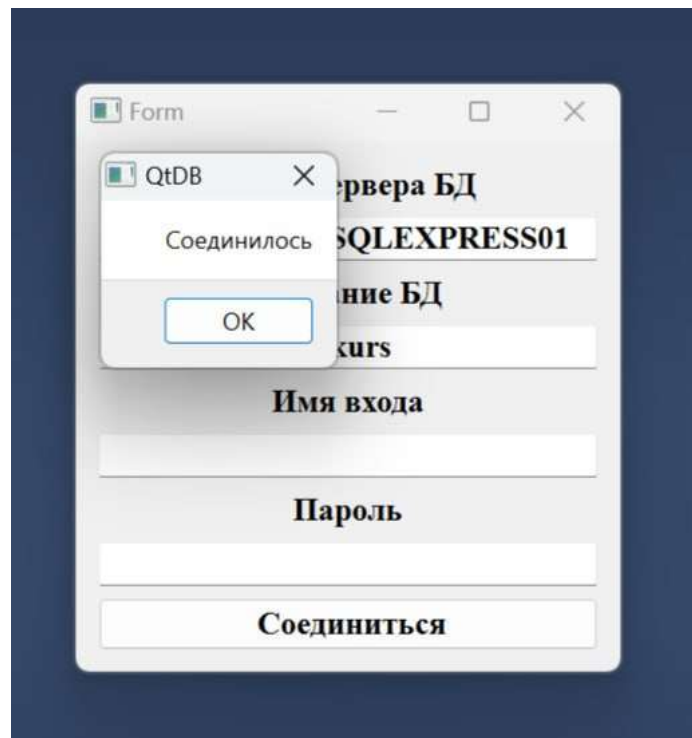


Рис.1.3 – Итог тестового подключения

Лабораторная работа №2. Отображение содержимого таблицы на главной форме. Добавление новой строки

1. На главную форму были добавлены объекты tableView и pushButton (Рис.2.1):

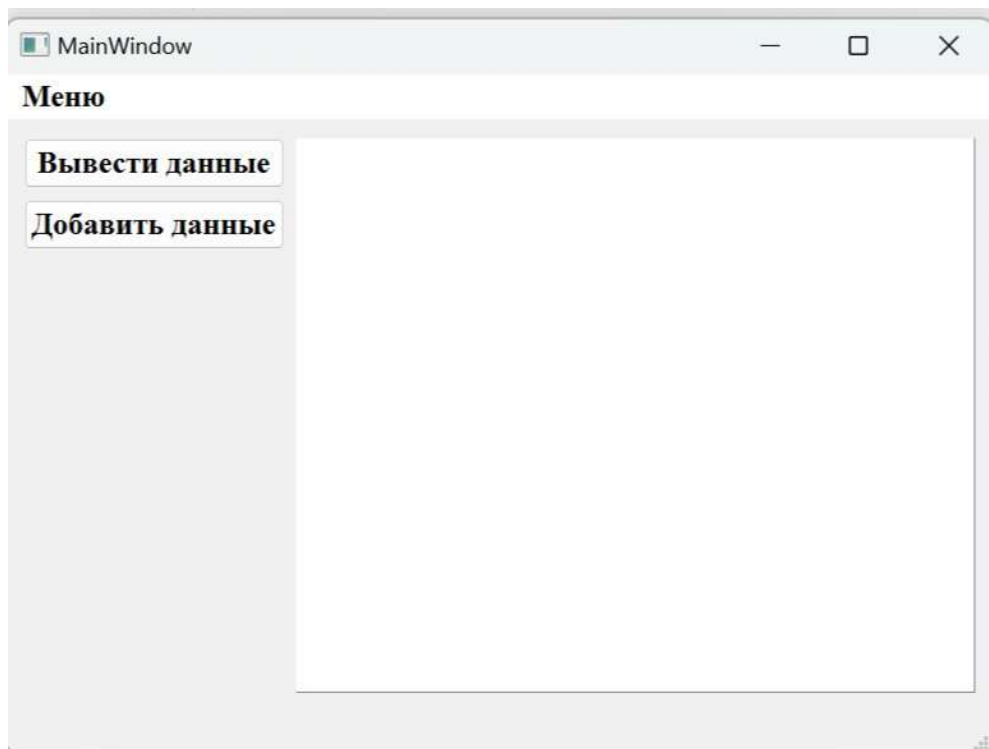


Рис.2.1 – Изменения главного окна

2. В данной лабораторной работе были добавлены следующие модели:
 - Модель QSqlTableModel — модель, оптимизированная для работы с одной таблицей, но предоставляет возможности выполнения операций чтениязаписи, которые можно производить из некоторых компонентов пользовательского интерфейса, например tableView.
 - Модель QSqlQueryModel — модель позволяющая выбирать данные из базы данных при помощи инструкции «SELECT», в котором могут быть использованы практически все предложения, поддерживаемые данной инструкцией (условия, группировка, сортировка и т.д.).

Для обновления экранной формы был описан метод обработчика нажатия на кнопку «Добавить данные» (Листинг 2.1):

Листинг 2.1 – Описание обработчика нажатия на кнопку «Добавить данные»

```
void MainWindow::on_pushButton_clicked()
{
    model = new QSqlQueryModel();
    model->setQuery("SELECT name,sort FROM primer");

    model->setHeaderData(0,Qt::Horizontal,"Имя");
    model->setHeaderData(1,Qt::Horizontal,"Сорт");

    ui->tableView->setModel(model);
    ui->tableView->resizeColumnsToContents();
    ui->tableView->show();
}
```

3. Обработчик кнопки «Добавить» был описан подобно окну “login” из ЛР №1, где обработчик кнопки на главном окне переводит нас на отдельное диалоговое окно, в котором мы добавляем строку в соответствии со следующим листингом:

Листинг 2.2 – Описание обработчика нажатия кнопки «Добавить данные» диалогового окна “addrecord”

```
void addrecord::on_pushButton_clicked()
{
    QSqlQuery* query = new QSqlQuery();
    query->prepare("INSERT INTO primer(name,sort) VALUES(:name,:sort)");
    query->bindValue("name",ui->lineEdit->text());
    query->bindValue("sort",ui->lineEdit_2->text());

    QMessageBox* mess = new QMessageBox();

    if(query->exec())
    {
        mess->setText("Запрос составлен неверно");
        mess->show();
    }

    emit refresh_table();
}
```

4. В итоге мы можем обновлять информацию о подключённой таблице и добавлять в неё новые строки, что видно на Рис.2.2-4:

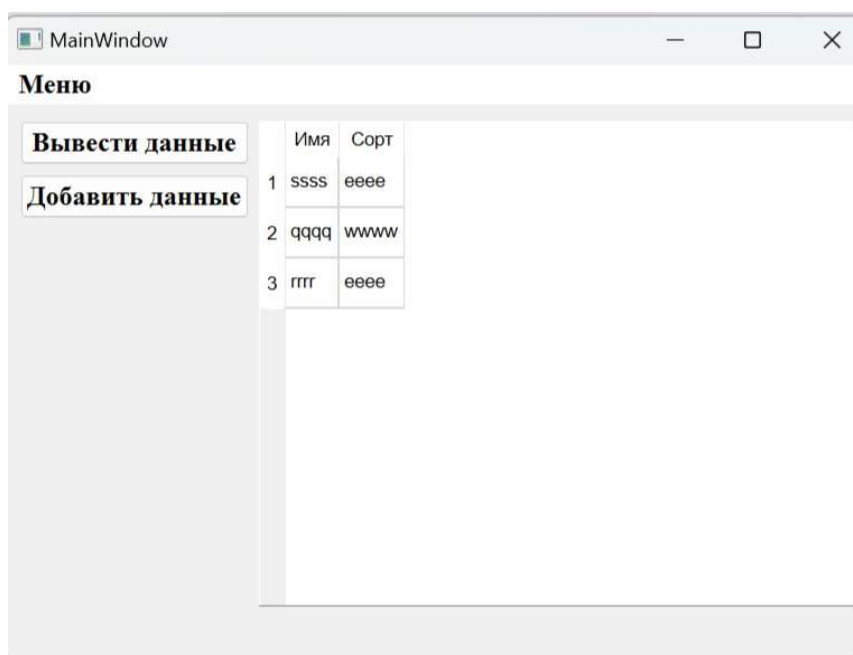


Рис.2.2 – Проверка кнопки «Вывести данные»

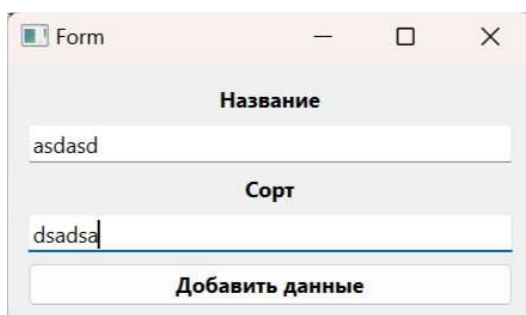


Рис.2.3 – Результат нажатия на кнопку «Добавить данные» на главном окне

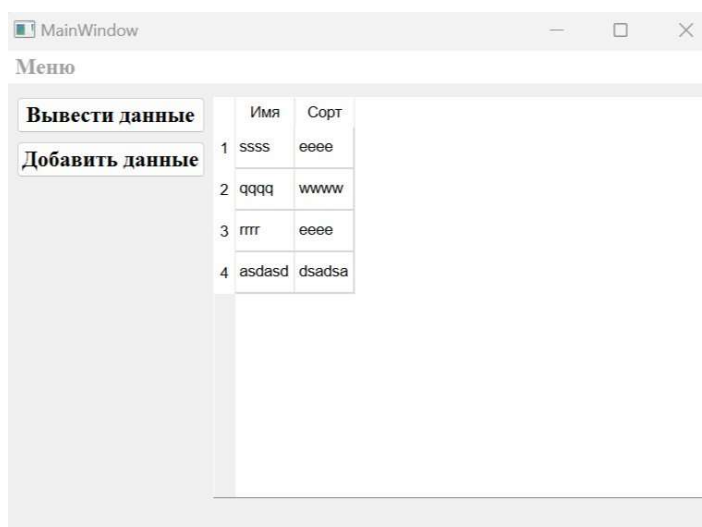


Рис.2.4 – Результат работы диалогового окна addrecord

Лабораторная работа №3. Изменение и удаление строк в таблице БД

1. В рамках лабораторной работы были добавлены элементы QLineEdit (имя и сорт) и QPushButton (изменить и удалить) как показано на Рис.3.1:

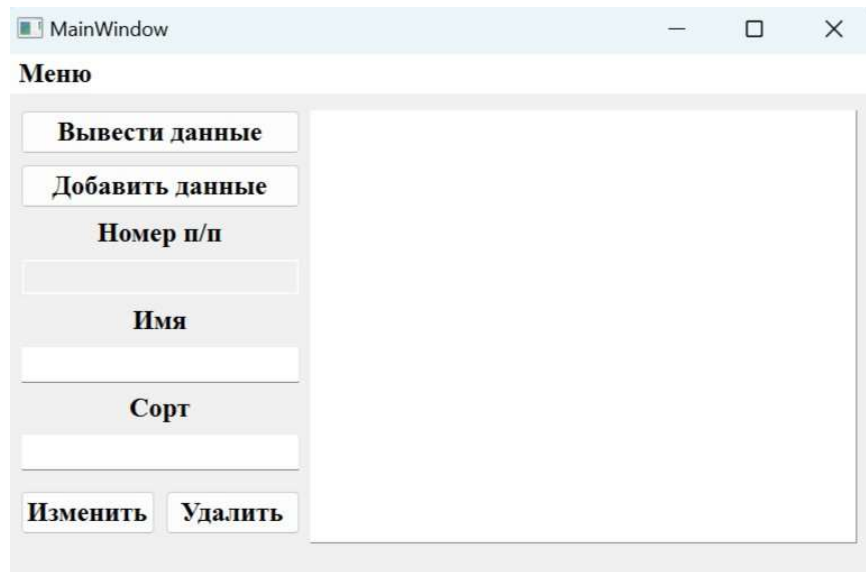


Рис.3.1 – Изменения главного диалогового окна

2. В файл mainwindow.cpp были добавлены методы CustomMenuReq(), DelRecAction() и ModRecAction() для дальнейшей работы с элементами для удаления и изменения строк таблицы.
3. Добавлен обработчик выбора строки в объекте tableView(Листинг 3.1):

Листинг 3.1 – Описание обработчика выбора строки таблицы

```
void MainWindow::on_tableView_clicked(const QModelIndex &index)
{
    int temp_nom;
    temp_nom =
ui->tableView->model()->data(ui->tableView->model()->index(index.row(),0)).toInt();
    ui->lineEdit->setText(QString::number(temp_nom));
    QSqlQuery* query = new QSqlQuery();
    query->prepare("SELECT name,sort FROM primer WHERE id=:id");
    query->bindValue(":id",temp_nom);
    if(query->exec())
    {
        query->next();
        ui->lineEdit_2->setText(query->value(0).toString());
        ui->lineEdit_3->setText(query->value(1).toString());
    }
}
```


4. Описан метод отработки нажатия на кнопку «Удалить» (Листинг 3.2).
Результат работы данного метода можно увидеть на Рис.3.2-3.3.

Листинг 3.2 – Метод отработки нажатия на кнопку «Удалить»

```
void MainWindow::on_pushButton_4_clicked()
{
    QSqlQuery* query = new QSqlQuery();
    query->prepare("DELETE FROM primer WHERE id=?");
    query->bindValue(0,ui->lineEdit->text());

    query->exec();
    on_pushButton_clicked();
}
```

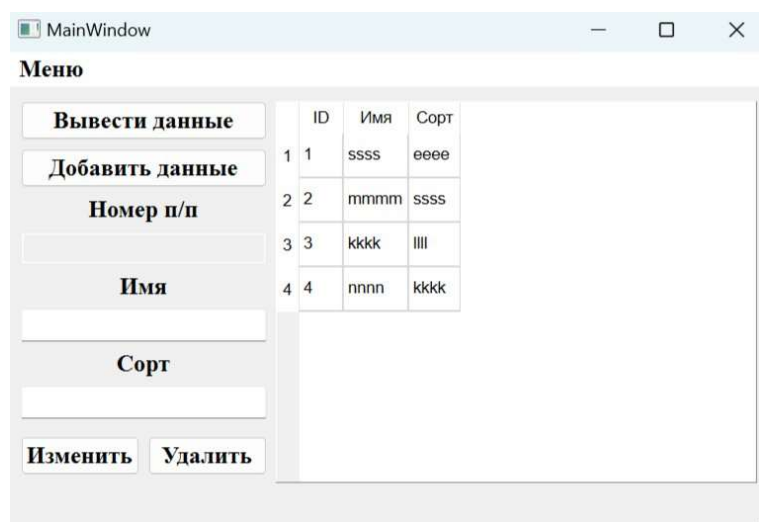


Рис.3.2 – Результат до нажатия на кнопку «Удалить»

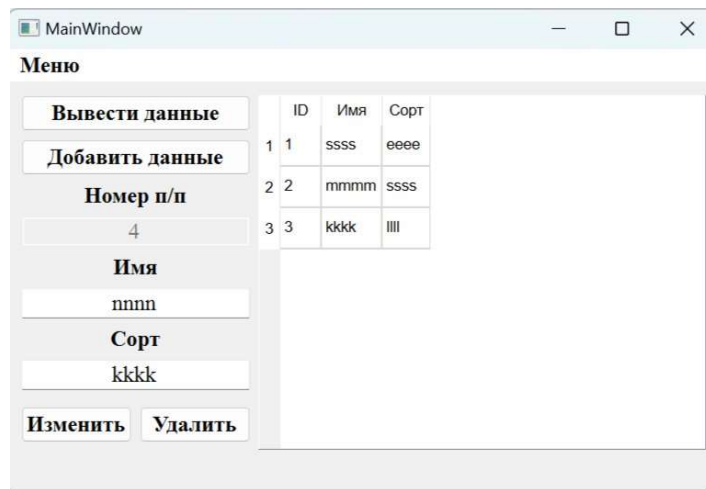


Рис.3.3 – Результат после нажатия на кнопку «Удалить»

5. Метод нажатия на кнопку «Изменить» был описан как в качестве кнопки на главном окне, так и в качестве отдельного диалогового окна,

которое появляется при нажатии на строку правой кнопкой мыши, метод обработки нажатия на кнопку на главном диалоговом окне можно увидеть на Листинге 3.3:

Листинг 3.3 – Обработчик нажатия на кнопку «Изменить» главного окна

```
void MainWindow::on_pushButton_3_clicked()
{
    QSqlQuery* query = new QSqlQuery();
    query->prepare("UPDATE primer SET name=?,sort=? WHERE id=?");
    query->bindValue(0,ui->lineEdit_2->text());
    query->bindValue(1,ui->lineEdit_3->text());
    query->bindValue(2,ui->lineEdit->text());

    query->exec();
    on_pushButton_clicked();
}
```

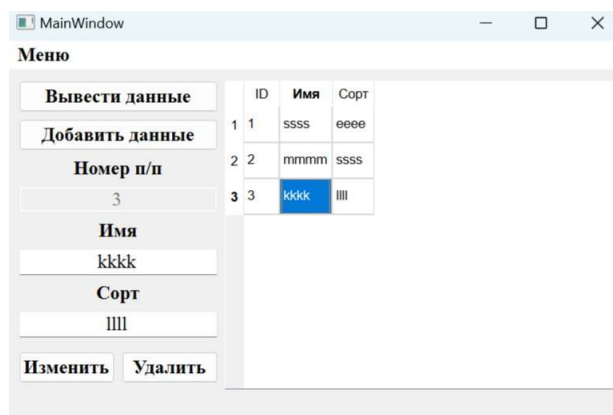


Рис.3.4 – Результат до работы кнопки «Изменить»

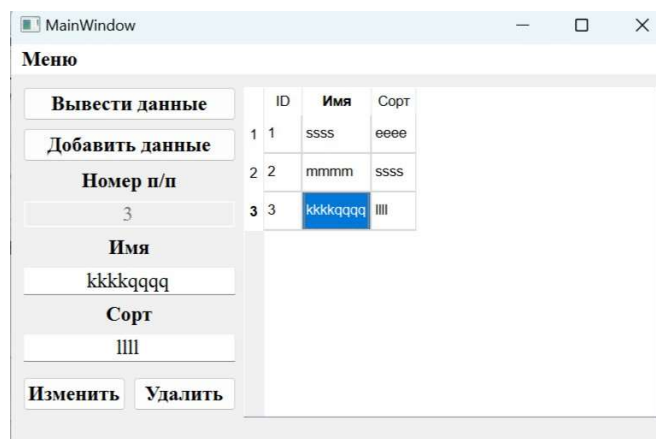


Рис.3.5 – Результат после работы кнопки «Изменить»

Лабораторная работа №4. Использование контекстного меню для изменения и удаления строк таблицы

1. Для создания контекстного меню был использован класс «QMenu».

Были описаны следующие методы контекстного меню (Листинг 4.1):

Листинг 4.1 – Метод контекстного меню

```
void MainWindow::CustomMenuReq(QPoint pos)
{
    if (fl==1)
    {
        QModelIndex index = ui->tableView->indexAt(pos);
        globid = ui->tableView->model()->data(ui->tableView->model()->index(index.row(),0)).toInt();
        QMenu* menu = new QMenu(this);
        QAction* izm = new QAction("Изменить",this);
        connect(izm,SIGNAL(triggered()),this,SLOT(izm_zap()));
        QAction* ud = new QAction("Удалить",this);
        connect(ud,SIGNAL(triggered()),this,SLOT(del_zap()));
        menu->addAction(izm);
        menu->addAction(ud);
        menu->popup(ui->tableView->viewport()->mapToGlobal(pos));
    }
}
```

```
void MainWindow::del_zap()
{
    QSqlQuery* query = new QSqlQuery();
    query->prepare("DELETE FROM primer WHERE id=?");
    query->bindValue(0,globalid);
    if (query->exec())
    {
        on_pushButton_clicked();
    }
}

void MainWindow::izm_zap()
{
    izm = new izmenenie();
    connect(this,SIGNAL(sendID(int)),izm,SLOT(obr_sendID(int)));
    emit sendID(globalid);
    izm->show();
    disconnect(this,SIGNAL(sendID(int)),izm,SLOT(obr_sendID(int)));
}
```

Лабораторная работа №5. Формирование отчёта

1. В проект были добавлены новые модули Qt: «axcontainer» и «printsupport». Также были подключены необходимые библиотеки для последующего вывода таблицы баз данных в формат «.doc» и «.pdf» (Листинг 5.1):

Листинг 5.1 – Подключение необходимых библиотек

```
#include <QPrinter>

#include <QTextDocument>

#include <QFileDialog>
```

Также была добавлена новая экранная форма для взаимодействия с нашей базой данных и выбора файла, в который мы будем импортировать данные (Рис.5.1):

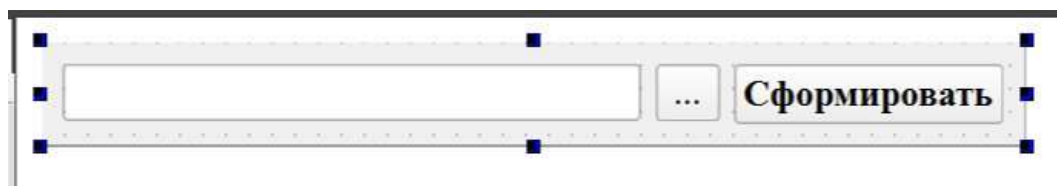


Рис.5.1 – Экранная форма формирования отчёта

2. После создания html-файла был создан метод копирования таблицы в файл MS Word в качестве описания отработки нажатия на кнопку «Сформировать» (Листинг 5.2):

Листинг 5.2 – Метод копирования таблицы в файл MS Word.

```
void print::on_pushButton_clicked()
{
    QFile* file = new QFile();
    file->setFileName(ui->lineEdit->text());
    file->open(QIODevice::WriteOnly);
    QTextStream in(file);
    in<<"<html><head></head><body><center>" + QString("Пример создания отчета");
    in<<"<table border=1><tr>";
    in<<"<td>" + QString("Номер") + "</td>";
    in<<"<td>" + QString("Название") + "</td>";
    in<<"<td>" + QString("Сорт") + "</td></tr>";
    QSqlQuery* query = new QSqlQuery();
    query->exec("SELECT id,name,sort FROM primer");
    while (query->next())
    {
        in<<"<tr><td>";
        in<<query->value(0).toString();
        in<<"</td><td>";
        in<<query->value(1).toString();
        in<<"</td><td>";
        in<<query->value(2).toString();
        in<<"</td></tr>";
    }
    in<<"</table></center></body></html>";
    file->close();
    QAxObject* word = new QAxObject("Word.Application",this);
    word->setProperty("DisplayAlerts",false);
    word->setProperty("Visible",true);
    QAxObject* doc = word->querySubObject("Documents");
    doc->dynamicCall("Open(QVariant)",ui->lineEdit->text());
}
```

Результатом после нажатия на кнопку «Сформировать» является отчёт в MS Word (Рис.5.1):

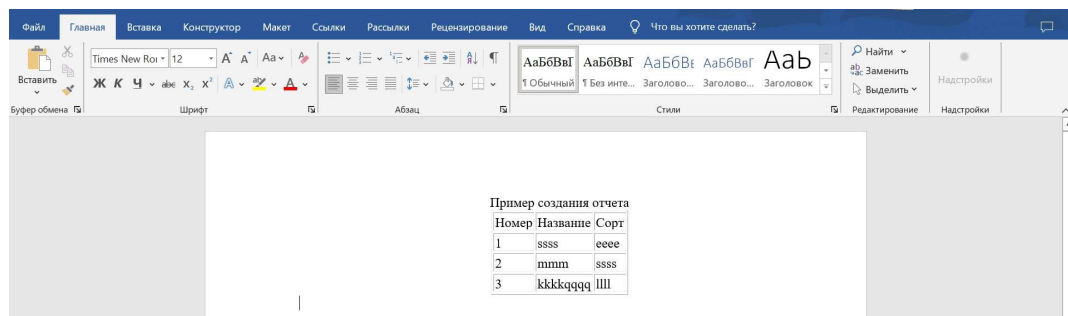


Рис.5.1 – Отработка нажатия на кнопку «Сформировать»

3. Метод, сохраняющий таблицу в PDF-файл описан следующим образом(Листинг 5.3):

Листинг 5.3. – Описание обработки кнопки «Печать в PDF»

```
void MainWindow::on_pushButton_6_clicked()
{
    QString str;
    str.append("<html><head></head><body><center>" + QString("Пример создания отчета"));
    str.append("<table border=1><tr>");
    str.append("<td>" + QString("Номер") + "</td>");
    str.append("<td>" + QString("Название") + "</td>");
    str.append("<td>" + QString("Сорт") + "</td></tr>");
    QSqlQuery* query = new QSqlQuery();
    query->exec("SELECT id,name,sort FROM primer");
    while (query->next())
    {
        str.append("<tr><td>");
        str.append(query->value(0).toString());
        str.append("</td><td>");
        str.append(query->value(1).toString());
        str.append("</td><td>");
        str.append(query->value(2).toString());
        str.append("</td></tr>");
    }
    str.append("</table></center></body></html>");
    QPrinter printer;
    printer.setPageOrientation(QPageLayout::Portrait);
    printer.setOutputFormat(QPrinter::PdfFormat);
    printer.setPageSize(QPageSize::A4);
    QString path = QFileDialog::getSaveFileName(NULL, "Сохранить PDF", "Отчет", "PDF (*.pdf)");
    if (path.isEmpty()) return;
    printer.setOutputFileName(path);
    QTextDocument doc;
    doc.setHtml(str);
    doc.print(&printer);
}
```

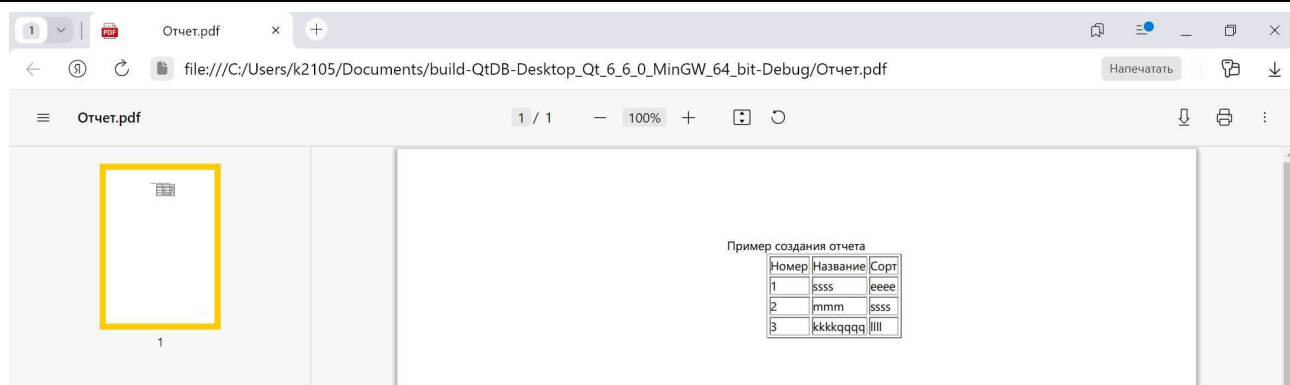
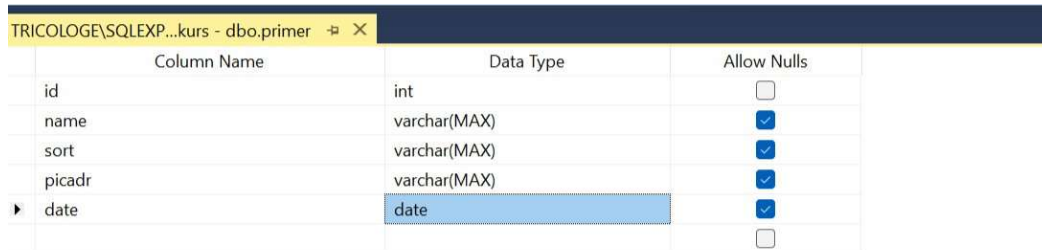


Рис.5.2 – Обработка кнопки «Экспорт в PDF»

Лабораторная работа №6. Добавление изображений и даты к записи

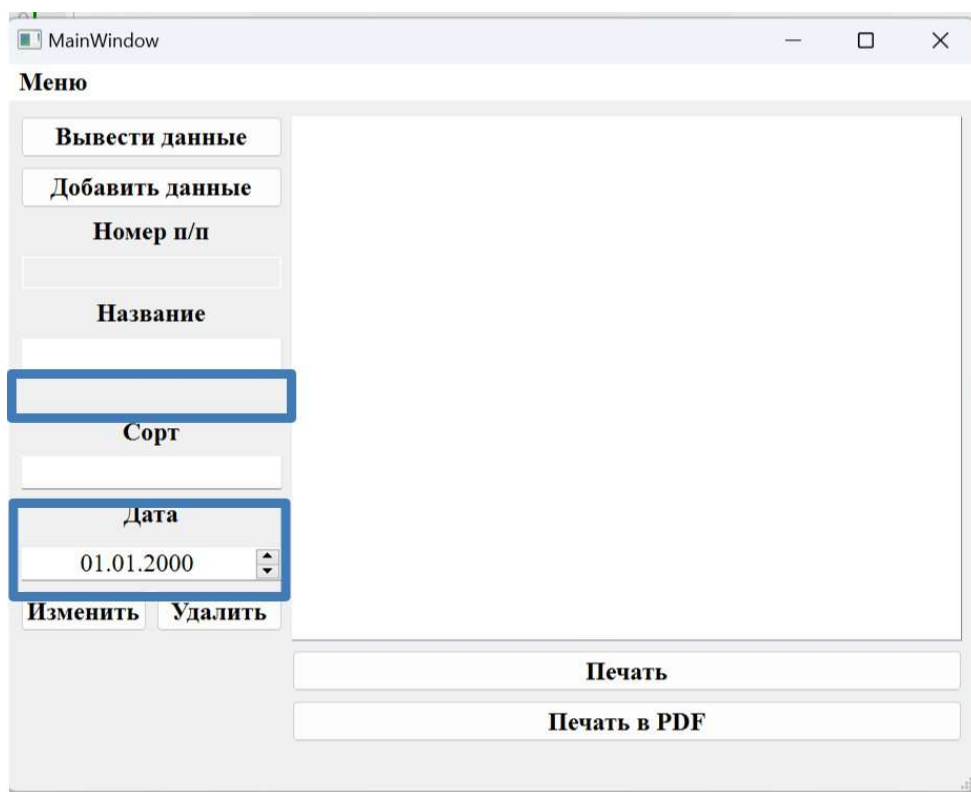
1. В таблицу primer были добавлены поля для хранения даты и пути изображения (Рис.6.1):



Column Name	Data Type	Allow Nulls
id	int	<input type="checkbox"/>
name	varchar(MAX)	<input checked="" type="checkbox"/>
sort	varchar(MAX)	<input checked="" type="checkbox"/>
picadr	varchar(MAX)	<input checked="" type="checkbox"/>
date	date	<input checked="" type="checkbox"/>

Рис.6.1 – Добавление столбцов picadr и date

2. На экранные формы mainwindow.ui и addrecord.ui были добавлены соответствующие кнопки для выбора изображения, а также объект dateEdit для выбора даты (Рис.6.2-6.3):



MainWindow

Меню

Вывести данные

Добавить данные

Номер п/п

Название

Сорт

Дата

01.01.2000

Изменить Удалить

Печать

Печать в PDF

Рис.6.2 – Итоговый вид окна mainwindow.ui

Рис.6.3 – Итоговый вид окна `addrecord.ui`

3. В метод `on_tableView_clicked()` были добавлены следующие строки для отображения изображения и даты при нажатии на таблицу левой кнопкой мыши:

Листинг 6.1 – Метод обработки нажатия на таблицу левой кнопкой мыши

```
void MainWindow::on_tableView_clicked(const QModelIndex &index)
{
    int temp_nom;
    temp_nom = ui->tableView->model()->data(ui->tableView->model()->index(index.row(),0)).toInt();
    ui->lineEdit->setText(QString::number(temp_nom));
    QSqlQuery* query = new QSqlQuery();
    query->prepare("SELECT name,sort,picadr,date FROM primer WHERE id=:id");
    query->bindValue(":id",temp_nom);
    if(query->exec())
    {
        query->next();
        ui->lineEdit_2->setText(query->value(0).toString());
        ui->lineEdit_3->setText(query->value(1).toString());
        ui->label_4->setScaledContents(true);
        ui->label_4->setPixmap(QPixmap(query->value(2).toString()));
        ui->dateEdit->setDate(QDate(query->value(3).toDate()));
    }
}
```

4. В методы добавления и изменения строки также были добавлены строки, необходимые для взаимодействия с таблицей согласно заданию(изменения в обоих случаях схожи, поэтому приводится только один Листинг):

Листинг 6.2 – Изменения метода отработки нажатия на кнопку «Добавить»

```
void addrecord::on_pushButton_clicked()
{
    count++;
    QSqlQuery* query = new QSqlQuery();
    query->prepare("INSERT INTO primer(id,name,sort,picadr,date) VALUES(:id,:name,:sort,:picadr,:date)");
    query->bindValue(":id",count);
    query->bindValue(":name",ui->lineEdit->text());
    query->bindValue(":sort",ui->lineEdit_2->text());
    query->bindValue(":picadr",imageadr);
    query->bindValue(":date",ui->dateEdit->text());
    QMessageBox* mess = new QMessageBox();
    if(!query->exec())
    {
        mess->setText("Запрос составлен неверно");
        mess->show();
    }
    emit refresh_table();
}
```

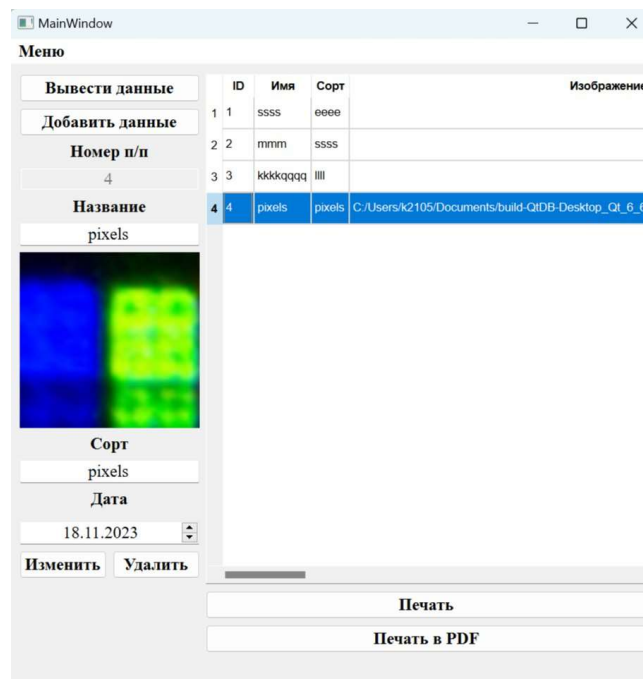



Рис.6.4 – Результат выполнения лабораторной работы

Лабораторная работа №7. Использование comboBox для выбора значения ячейки

1. В результате выполнения скрипта, приведенного в приложении, получилась следующая таблица postable, которая использовалась дальше:



	id	sort
1	1	asddas
2	2	aaggtgr
3	3	afghrfesd
4	4	okdpoqkwd
5	5	asdasd

Рис.7.1 – Результат выполнения скрипта в SSMS

2. На всех диалоговых окнах объект lineEdit для выбора и изменения категории был заменён на comboBox:

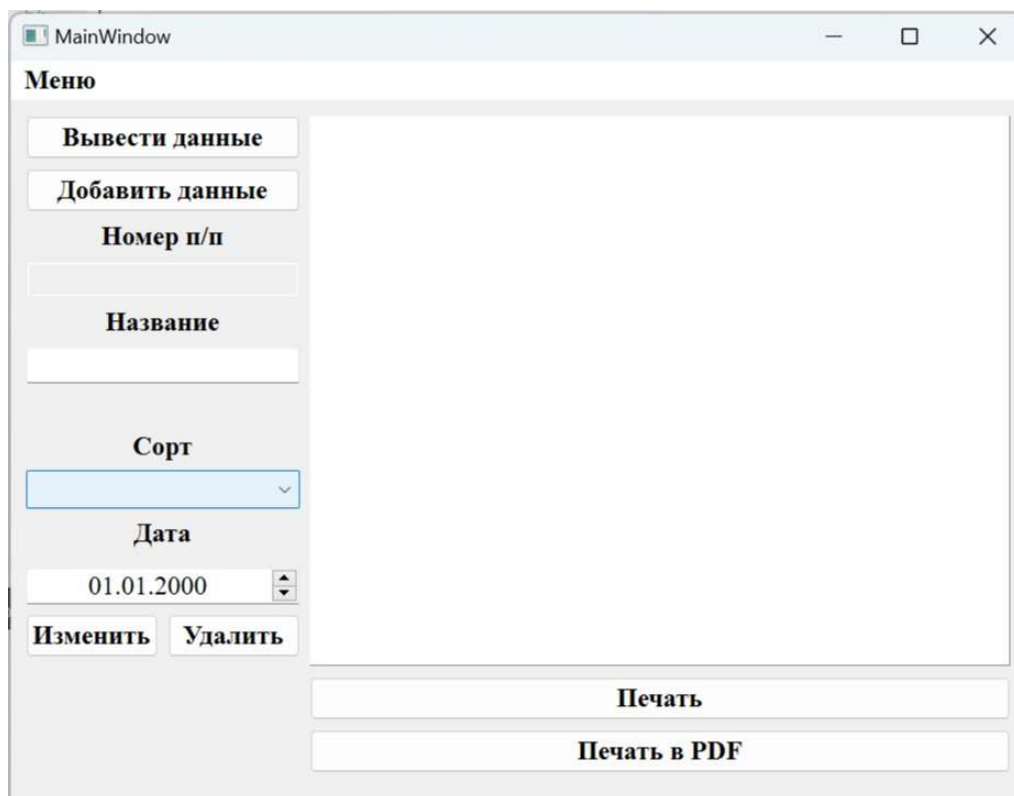


Рис.7.2 – Изменение объекта lineEdit в окне mainwindow.ui

3. Для того, чтобы список возможных категорий сразу выводился в окне, в метод отработки нажатия на кнопку «Обновить» были добавлены следующие строки:

Листинг.7.1 – Изменения метода обработки нажатия на кнопку «Обновить»

```
void MainWindow::on_pushButton_clicked()
{
    fl=1;
    ui->comboBox->clear();
    model = new QSqlQueryModel();
    model->setQuery("SELECT * FROM primer");

    model->setHeaderData(0,Qt::Horizontal,"ID");
    model->setHeaderData(1,Qt::Horizontal,"Имя");
    model->setHeaderData(2,Qt::Horizontal,"Сорт");
    model->setHeaderData(3,Qt::Horizontal,"Изображение");
    model->setHeaderData(4,Qt::Horizontal,"Дата");

    ui->tableView->setModel(model);
    ui->tableView->resizeColumnsToContents();
    ui->tableView->show();

    QSqlQuery* query = new QSqlQuery();
    query->exec("SELECT sort FROM postable");
    while (query->next())
    {
        ui->comboBox->addItem(query->value(0).toString());
    }
    sortcombo = 0;
}
```

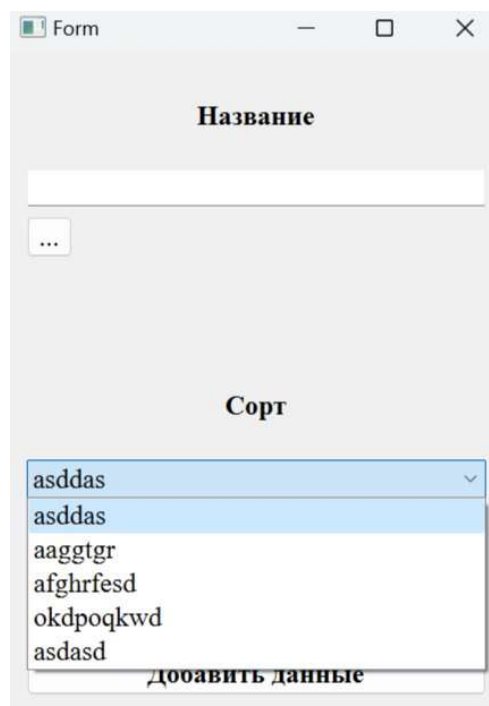


Рис.7.3 – Результат выполнения лабораторной работы

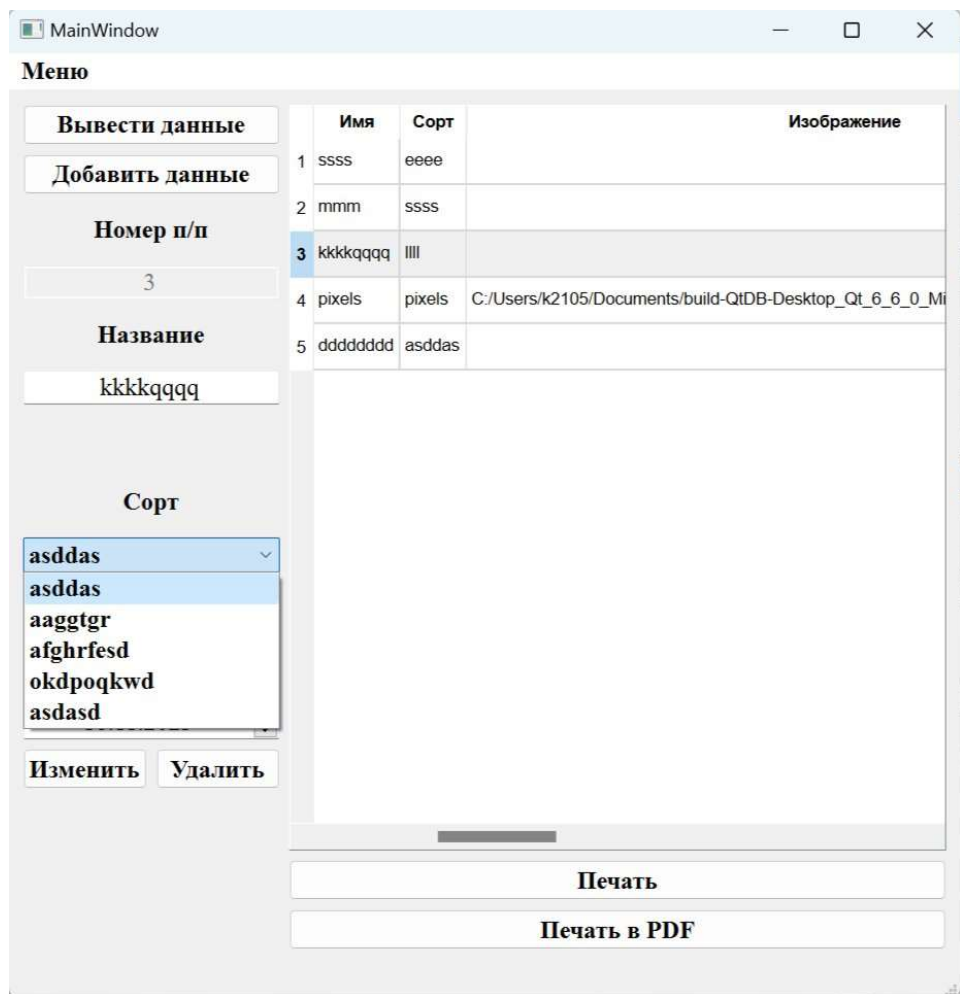


Рис.7.4 – Отображение категорий в главном диалоговом окне

Лабораторная работа №8. Создание графика и сохранение его в PDF

1. Для последующей работы с графиком была создана таблица chart со следующими строками:

Results		Messages
	id	score
1	1	3
2	2	5
3	3	10
4	4	1
5	5	-3
6	6	0
7	7	5
8	8	20
9	9	15
10	10	13
11	11	7
12	12	3

Рис.8.1 – Таблица chart

2. Была добавлена новая экранная форма “printgraph”, на которой были расположены компоненты widget и gridlayout:

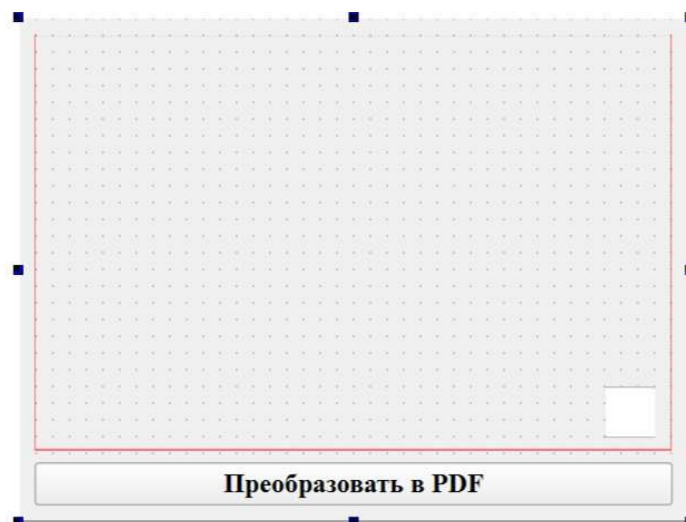


Рис.8.2 – Экранная форма “printgraph”

3. Для реализация метода был использован класс QCustomPlot, объекты которого позволяют отрисовать граф (Листинг 8.1):

Листинг 8.1 – Отображение графика по таблице chart в компоненте widget

```

printgraph::printgraph(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::printgraph) {
    ui->setupUi(this);
    QCPDocumentObject* poHandler = new QCPDocumentObject(this);
    ui->textEdit->document()->documentLayout()->registerHandler(QCPDocumentObject::PlotTextFormat,poHandler);
    ui->textEdit->setVisible(false);
    ui->widget->plotLayout()->insertRow(0);
    ui->widget->plotLayout()->addElement(0,0,new QCPTextElement(ui->widget,"График выручки"));
    QVector<double> dx,dy;
    double minX,minY,maxX,maxY;
    minX=0;
    minY=0;
    maxX=0;
    maxY=0;
    QSqlQuery* query = new QSqlQuery();
    if (query->exec("SELECT * FROM chart")) {
        while(query->next()) {
            if (minX>=query->value(0).toDouble()) minX = query->value(0).toDouble();
            if (minY>=query->value(1).toDouble()) minY = query->value(1).toDouble();
            if (maxX<=query->value(0).toDouble()) maxX = query->value(0).toDouble();
            if (maxY<=query->value(1).toDouble()) maxY = query->value(1).toDouble();
            dx << query->value(0).toDouble();
            dy << query->value(1).toDouble();
            QCPBars* bar = new QCPBars(ui->widget->xAxis,ui->widget->yAxis);
            bar->setName("Значение");
            bar->setBrush(QColor(255,0,0,255));
            bar->setData(dx,dy);
            bar->setWidth(0.20);
            ui->widget->xAxis->setLabel("Месяц");
            ui->widget->yAxis->setLabel("Выручка (млн.)");
            ui->widget->xAxis->setRange(minX,maxX+0.20);
            ui->widget->yAxis->setRange(minY,maxY+1);
            QSharedPointer<QCPAxisTickerFixed> fixedTicker(new QCPAxisTickerFixed);
            ui->widget->xAxis->setTicker(fixedTicker);
            ui->widget->yAxis->setTicker(fixedTicker);
            fixedTicker->setTickStep(1.0);
            fixedTicker->setScaleStrategy(QCPAxisTickerFixed::ssNone);
            ui->widget->replot();
        }
    }
    QTextCursor cur = ui->textEdit->textCursor();
    cur.inse
    rtText(QString(
    QChar::ObjectReplacementCh
    aracter),QCPDocumentObject::generatePlotFormat(ui->widg
    et,480,340));
}

```


4. Для сохранения файла в формате PDF был описан метод обработки нажатия на кнопку «Экспортировать в PDF» диалогового окна printgraf.ui:

Листинг 8.2 – Метод обработки нажатия на кнопку «Экспортировать в PDF»

```
void printgraph::on_pushButton_clicked()
{
    QString fn = QFileDialog::getSaveFileName(0, "Сохранить в PDF", ".", "*.pdf");
    if (!fn.isEmpty())
    {
        QPrinter print;
        print.setFullPage(true);
        print.setPageSize(QPageSize::A4);
        print.setPageOrientation(QPageLayout::Portrait);
        print.setOutputFormat(QPrinter::PdfFormat);
        print.setOutputFileName(fn);
        ui->textEdit->document()->print(&print);
    }
}
```

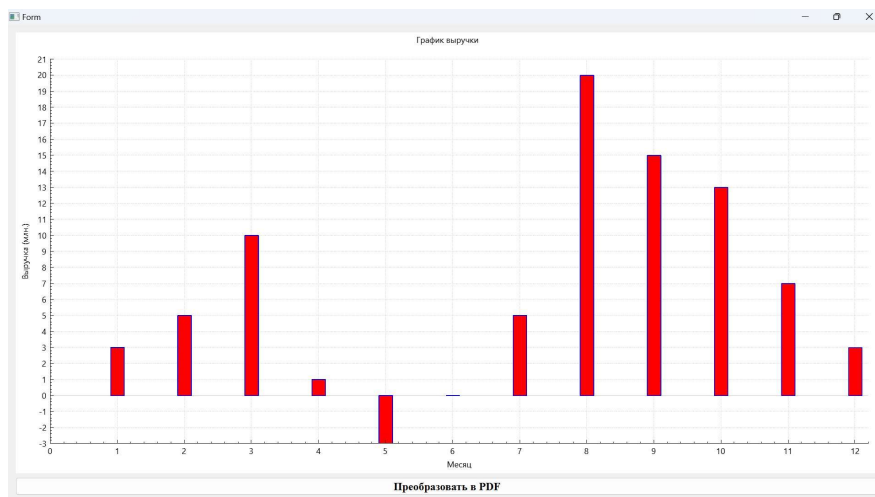


Рис.8.3 – Отрисовка графика в диалоговом окне



Рис.8.4 – Открытый PDF-файл после сохранения