Министерство образования Республики Беларусь  
Учреждение образования  
«Брестский государственный технический университет»  
Кафедра ИИТ

Лабораторная работа №6, 7, 8  
По дисциплине: «Создание приложения для работы с БД организация пользовательского интерфейса: табличное представление и отчеты»

Выполнил:  
студент 3 курса  
группы ИИ-24  
Крейдич А.А.

Проверил:  
Козик И. Д.

Брест 2025

**Цель работы**: получить навыки разработки приложений БД и организации пользовательского интерфейса: табличного представления информации и построения отчетов.

**Задание**

Изучить материал, приведенный в “К лабораторной работе 6.doc”.

На основании логической модели (в соответствии с вариантом Лабораторной работы №1) создать приложение и организовать пользовательский интерфейс, используя соответствующие элементы и методы: организовать табличное представление информации и построить отчеты не менее 4.

**Код программы:**

import tkinter as tk

from tkinter import ttk, messagebox

import sqlite3

import re

import hashlib

from tkcalendar import DateEntry

class DatabaseManager:

    """

    Класс-обертка для работы с базой данных.

    Инкапсулирует подключение, выполнение запросов и получение данных.

    """

    def \_\_init\_\_(self, connection\_params):

        self.connection\_params = connection\_params

        self.conn = sqlite3.connect(\*\*connection\_params)

        self.cursor = self.conn.cursor()

    def get\_table\_names(self):

        self.cursor.execute("SELECT name FROM sqlite\_master WHERE type='table';")

        return [row[0] for row in self.cursor.fetchall()]

    def execute(self, query, params=()):

        try:

            self.cursor.execute(query, params)

            self.conn.commit()

        except sqlite3.Error as e:

            messagebox.showerror("Ошибка БД", f"Произошла ошибка: {e}")

            return False

        return True

    def fetchall(self, query, params=()):

        self.cursor.execute(query, params)

        return self.cursor.fetchall()

    def fetchone(self, query, params=()):

        self.cursor.execute(query, params)

        return self.cursor.fetchone()

    def close(self):

        self.conn.close()

class DatabaseApp:

    """

    Основной класс приложения. Отвечает за интерфейс,

    работу с виджетами и взаимодействие с базой данных через DatabaseManager.

    """

    def \_\_init\_\_(self, master, connection\_params):

        self.master = master

        self.master.title("Военкомат")

        self.master.configure(bg="#f0f0f0")

        self.db = DatabaseManager(connection\_params)

        self.setup\_styles()

        self.create\_header()

        self.create\_report\_button()

        self.notebook = ttk.Notebook(master)

        self.notebook.pack(expand=True, fill='both', padx=10, pady=10)

        self.table\_names = self.db.get\_table\_names()

        for table\_name in self.table\_names:

            frame = tk.Frame(self.notebook, bg="#f0f0f0")

            self.notebook.add(frame, text=table\_name)

            self.create\_table\_view(frame, table\_name)

    def setup\_styles(self):

        """Настройка стилей для виджетов приложения."""

        style = ttk.Style(self.master)

        style.theme\_use("clam")

        style.configure("TButton", font=("Arial", 10, "bold"), padding=6, background="#4CAF50", foreground="white")

        style.map("TButton", background=[("active", "#45a049")])

        style.configure("TLabel", font=("Arial", 10), background="#f0f0f0", foreground="#333")

        style.configure("TNotebook", background="#f0f0f0")

        style.configure("TNotebook.Tab", font=("Arial", 10, "bold"), padding=[10, 5])

        style.configure("Treeview", font=("Arial", 10), rowheight=25)

        style.configure("Treeview.Heading", font=("Arial", 10, "bold"))

    def create\_header(self):

        """Создание заголовка приложения."""

        header\_label = tk.Label(self.master, text="Система управления военкоматом",

                                font=("Arial", 16, "bold"), bg="#f0f0f0", fg="#333")

        header\_label.pack(side=tk.TOP, fill=tk.X, pady=(10, 5))

    def create\_report\_button(self):

        """Создание кнопки формирования отчёта."""

        report\_button = ttk.Button(self.master, text="Создать отчёт", command=self.generate\_report)

        report\_button.pack(side=tk.TOP, padx=10, pady=5)

    def create\_table\_view(self, frame, table\_name):

        """

        Создает представление для таблицы базы данных, включая Treeview

        и кнопки для действий (добавить, удалить, изменить, обновить).

        """

        columns\_info = self.db.fetchall(f"PRAGMA table\_info('{table\_name}');")

        columns = [col[1] for col in columns\_info]

        tree = ttk.Treeview(frame, columns=columns, show='headings', selectmode='browse')

        tree.pack(expand=True, fill='both', padx=5, pady=5)

        for col in columns:

            tree.heading(col, text=col)

            tree.column(col, width=100, anchor='center')

        self.populate\_treeview(tree, table\_name)

        btn\_frame = tk.Frame(frame, bg="#f0f0f0")

        btn\_frame.pack(pady=5)

        ttk.Button(btn\_frame, text="Добавить",

                   command=lambda: self.open\_row\_dialog(tree, table\_name, mode="add")).grid(row=0, column=0, padx=5)

        ttk.Button(btn\_frame, text="Удалить",

                   command=lambda: self.delete\_row(tree, table\_name)).grid(row=0, column=1, padx=5)

        ttk.Button(btn\_frame, text="Изменить",

                   command=lambda: self.open\_row\_dialog(tree, table\_name, mode="edit")).grid(row=0, column=2, padx=5)

        ttk.Button(btn\_frame, text="Обновить",

                   command=lambda: self.populate\_treeview(tree, table\_name)).grid(row=0, column=3, padx=5)

    def populate\_treeview(self, tree, table\_name):

        """Заполнение Treeview данными из таблицы."""

        data = self.db.fetchall(f"SELECT \* FROM '{table\_name}';")

        tree.delete(\*tree.get\_children())

        for row in data:

            tree.insert('', 'end', values=row)

    def validate\_and\_transform(self, table\_name, columns, values):

        """

        Валидация и преобразование введенных данных для каждой таблицы.

        Например, проверка формата номера телефона или преобразование значения из комбобокса.

        """

        new\_values = list(values)

        if table\_name == "Граждане":

            if "Телефон" in columns:

                index = columns.index("Телефон")

                phone = new\_values[index]

                phone\_digits = re.sub(r'\D', '', phone)

                if len(phone\_digits) != 11:

                    messagebox.showerror("Ошибка", "Номер телефона должен содержать 11 цифр.")

                    return None

                new\_values[index] = f"+{phone\_digits[0]}-{phone\_digits[1:4]}-{phone\_digits[4:7]}-{phone\_digits[7:9]}-{phone\_digits[9:11]}"

        elif table\_name in ("Призывники", "Документы", "Отсрочки"):

            if "Гражданин\_id" in columns:

                index = columns.index("Гражданин\_id")

                citizen\_val = new\_values[index]

                try:

                    if ":" in citizen\_val:

                        new\_values[index] = int(citizen\_val.split(":")[0].strip())

                    else:

                        new\_values[index] = int(citizen\_val)

                except ValueError:

                    messagebox.showerror("Ошибка", "Неверный формат id гражданина.")

                    return None

        return new\_values

    def open\_row\_dialog(self, tree, table\_name, mode="add"):

        """

        Универсальный диалог для добавления/редактирования записи.

        Если mode == "edit", предварительно заполняются текущие данные выбранной строки.

        """

        is\_edit = (mode == "edit")

        if is\_edit:

            selected = tree.selection()

            if not selected:

                messagebox.showwarning("Предупреждение", "Пожалуйста, выберите строку для изменения.")

                return

            current\_values = tree.item(selected)['values']

        else:

            current\_values = None

        columns\_info = self.db.fetchall(f"PRAGMA table\_info('{table\_name}');")

        columns = [col[1] for col in columns\_info]

        dialog = tk.Toplevel(self.master)

        dialog.title("Изменить строку" if is\_edit else "Добавить строку")

        dialog.configure(bg="#f0f0f0")

        entry\_widgets = []

        for i, col in enumerate(columns):

            ttk.Label(dialog, text=col).grid(row=i, column=0, padx=10, pady=5, sticky='e')

            widget = None

            # Для полей, связанных с гражданами, используем комбобокс

            if table\_name in ("Призывники", "Документы", "Отсрочки") and col == "Гражданин\_id":

                citizens = self.db.fetchall("SELECT id, ФИО FROM 'Граждане';")

                citizen\_values = [f"{c[0]}: {c[1]}" for c in citizens]

                widget = ttk.Combobox(dialog, state="readonly", values=citizen\_values)

                if is\_edit and current\_values:

                    try:

                        citizen\_name = self.db.fetchone("SELECT ФИО FROM 'Граждане' WHERE id = ?;", (current\_values[i],))[0]

                        widget.set(f"{current\_values[i]}: {citizen\_name}")

                    except Exception:

                        widget.set(str(current\_values[i]))

            elif table\_name == "Граждане" and col == "Дата\_рождения":

                widget = DateEntry(dialog, date\_pattern='dd.mm.yyyy')

                if is\_edit and current\_values:

                    widget.set\_date(current\_values[i])

            elif table\_name == "Призывники" and col == "Дата\_призыва":

                widget = DateEntry(dialog, date\_pattern='dd.mm.yyyy')

                if is\_edit and current\_values:

                    widget.set\_date(current\_values[i])

            elif table\_name == "Документы" and col == "Дата\_выдачи":

                widget = DateEntry(dialog, date\_pattern='dd.mm.yyyy')

                if is\_edit and current\_values:

                    widget.set\_date(current\_values[i])

            elif table\_name == "Отсрочки" and col in ("Дата\_выдачи", "Срок\_действия"):

                widget = DateEntry(dialog, date\_pattern='dd.mm.yyyy')

                if is\_edit and current\_values:

                    widget.set\_date(current\_values[i])

            else:

                widget = ttk.Entry(dialog)

                if is\_edit and current\_values:

                    widget.insert(0, current\_values[i])

            widget.grid(row=i, column=1, padx=10, pady=5, sticky='w')

            entry\_widgets.append(widget)

        def on\_submit():

            new\_values = [w.get() for w in entry\_widgets]

            validated = self.validate\_and\_transform(table\_name, columns, new\_values)

            if validated is None:

                return

            if is\_edit:

                # Формирование запроса для обновления записи

                set\_clause = ', '.join([f"{col} = ?" for col in columns])

                where\_clause = ' AND '.join([f"{col} = ?" for col in columns])

                query = f"UPDATE '{table\_name}' SET {set\_clause} WHERE {where\_clause};"

                if self.db.execute(query, validated + current\_values):

                    self.populate\_treeview(tree, table\_name)

                    dialog.destroy()

            else:

                placeholders = ', '.join(['?' for \_ in validated])

                query = f"INSERT INTO '{table\_name}' VALUES ({placeholders});"

                if self.db.execute(query, validated):

                    self.populate\_treeview(tree, table\_name)

                    dialog.destroy()

        ttk.Button(dialog, text="Подтвердить", command=on\_submit).grid(row=len(columns), columnspan=2, pady=10)

    def delete\_row(self, tree, table\_name):

        """Удаление выбранной записи с подтверждением."""

        selected = tree.selection()

        if not selected:

            messagebox.showwarning("Предупреждение", "Пожалуйста, выберите строку для удаления.")

            return

        confirm = messagebox.askyesno("Подтверждение", "Вы уверены, что хотите удалить эту строку?")

        if not confirm:

            return

        values = tree.item(selected)['values']

        columns = tree['columns']

        where\_clause = ' AND '.join([f"{col} = ?" for col in columns])

        query = f"DELETE FROM '{table\_name}' WHERE {where\_clause};"

        if self.db.execute(query, values):

            self.populate\_treeview(tree, table\_name)

    def generate\_report(self):

        """Формирование отчёта по базе данных и сохранение его в файл report.txt."""

        report\_window = tk.Toplevel(self.master)

        report\_window.title("Отчёт по базе данных")

        report\_window.configure(bg="#f0f0f0")

        text\_widget = tk.Text(report\_window, wrap='word', width=100, height=30, font=("Arial", 10))

        text\_widget.pack(expand=True, fill='both', padx=10, pady=10)

        scrollbar = tk.Scrollbar(report\_window, command=text\_widget.yview)

        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        text\_widget.config(yscrollcommand=scrollbar.set)

        report\_lines = []

        report\_lines.append("Отчёт по базе данных\n")

        report\_lines.append(f"База данных: {self.db.connection\_params.get('database')}\n")

        report\_lines.append("=" \* 80 + "\n\n")

        for table in self.table\_names:

            report\_lines.append(f"Таблица: {table}\n")

            columns\_info = self.db.fetchall(f"PRAGMA table\_info('{table}');")

            columns = [col[1] for col in columns\_info]

            report\_lines.append("Столбцы: " + ", ".join(columns) + "\n")

            count = self.db.fetchone(f"SELECT COUNT(\*) FROM '{table}';")[0]

            report\_lines.append(f"Количество записей: {count}\n")

            sample\_rows = self.db.fetchall(f"SELECT \* FROM '{table}' LIMIT 5;")

            if sample\_rows:

                report\_lines.append("Примеры записей:\n")

                for row in sample\_rows:

                    report\_lines.append(" | ".join([str(item) for item in row]) + "\n")

            else:

                report\_lines.append("Записей нет.\n")

            report\_lines.append("-" \* 80 + "\n\n")

        text\_widget.insert("1.0", "".join(report\_lines))

        text\_widget.config(state="disabled")

        with open("report.txt", "w", encoding="utf-8") as f:

            f.write("".join(report\_lines))

if \_\_name\_\_ == "\_\_main\_\_":

    connection\_params = {"database": "military\_draft.sqlite3"}

    # Создаем таблицы, если они отсутствуют

    conn = sqlite3.connect(\*\*connection\_params)

    cursor = conn.cursor()

    cursor.execute('''

    CREATE TABLE IF NOT EXISTS "Граждане" (

        "id" INTEGER PRIMARY KEY AUTOINCREMENT,

        "ФИО" TEXT NOT NULL,

        "Дата\_рождения" TEXT,

        "Адрес" TEXT,

        "Телефон" TEXT,

        "Email" TEXT

    );

    ''')

    cursor.execute('''

    CREATE TABLE IF NOT EXISTS "Призывники" (

        "id" INTEGER PRIMARY KEY AUTOINCREMENT,

        "Гражданин\_id" INTEGER,

        "Дата\_призыва" TEXT,

        "Статус" TEXT,

        FOREIGN KEY("Гражданин\_id") REFERENCES "Граждане"("id")

    );

    ''')

    cursor.execute('''

    CREATE TABLE IF NOT EXISTS "Документы" (

        "id" INTEGER PRIMARY KEY AUTOINCREMENT,

        "Гражданин\_id" INTEGER,

        "Тип\_документа" TEXT,

        "Номер" TEXT,

        "Дата\_выдачи" TEXT,

        FOREIGN KEY("Гражданин\_id") REFERENCES "Граждане"("id")

    );

    ''')

    cursor.execute('''

    CREATE TABLE IF NOT EXISTS "Сотрудники" (

        "id" INTEGER PRIMARY KEY AUTOINCREMENT,

        "ФИО" TEXT NOT NULL,

        "Должность" TEXT,

        "Отдел" TEXT

    );

    ''')

    cursor.execute('''

    CREATE TABLE IF NOT EXISTS "Отсрочки" (

        "id" INTEGER PRIMARY KEY AUTOINCREMENT,

        "Гражданин\_id" INTEGER,

        "Причина" TEXT,

        "Дата\_выдачи" TEXT,

        "Срок\_действия" TEXT,

        FOREIGN KEY("Гражданин\_id") REFERENCES "Граждане"("id")

    );

    ''')

    conn.commit()

    conn.close()

    try:

        root = tk.Tk()

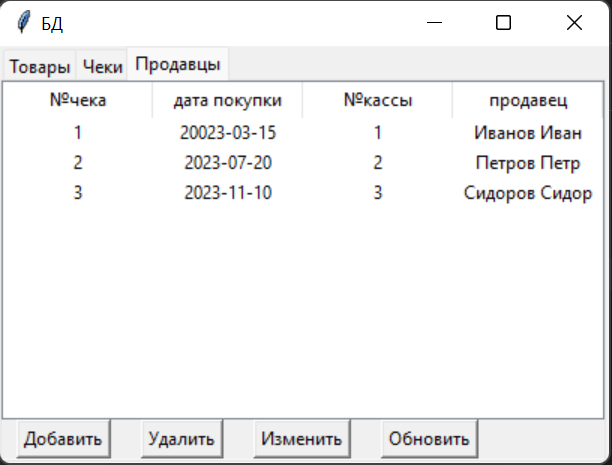
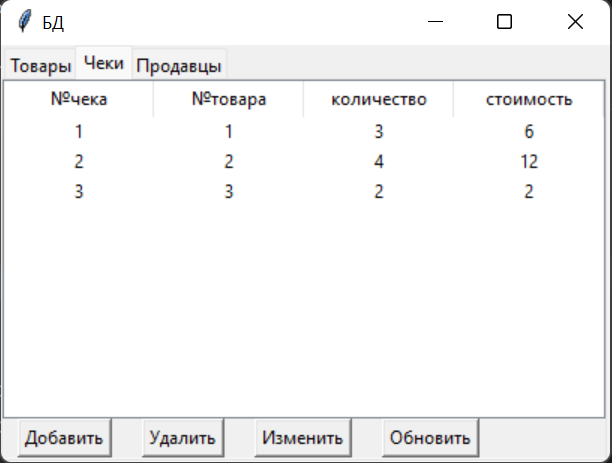
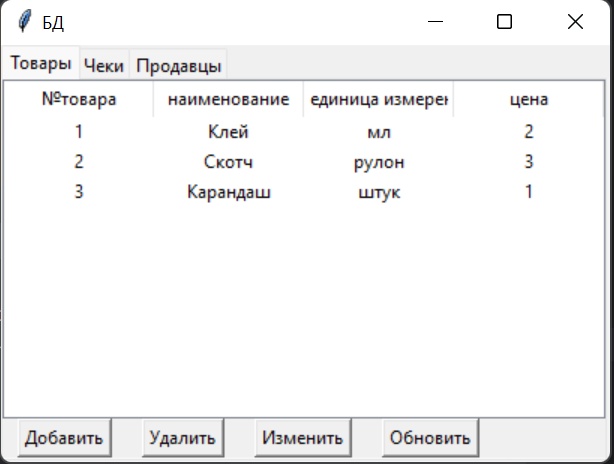
        app = DatabaseApp(root, connection\_params)

        root.mainloop()

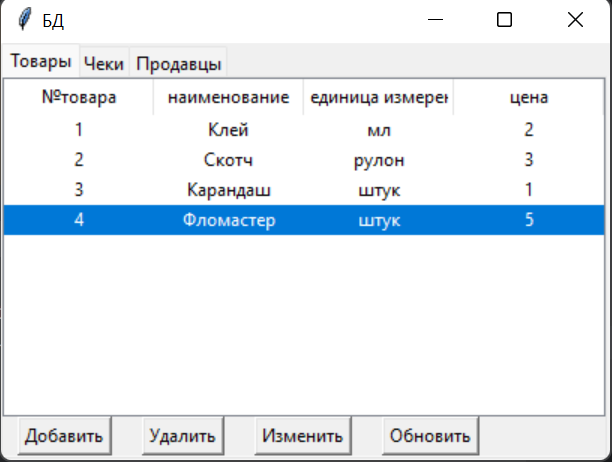
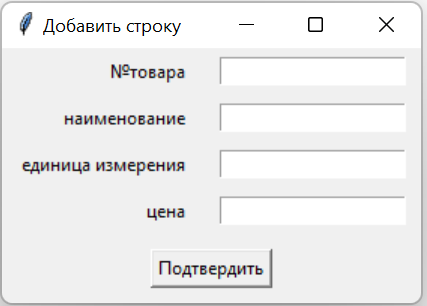
    except sqlite3.Error as err:

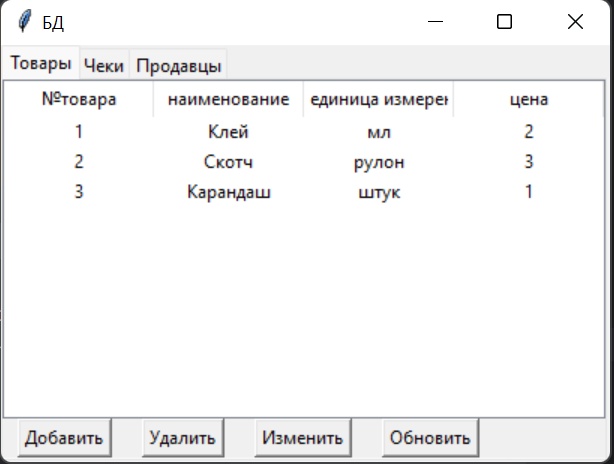
        print(f"Error: {err}")

**Таблицы:**

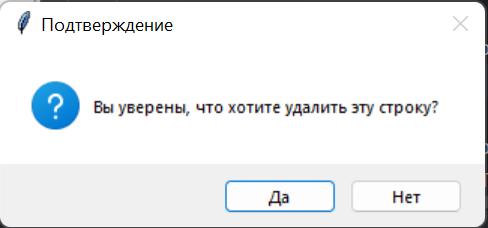


**Добавление:**

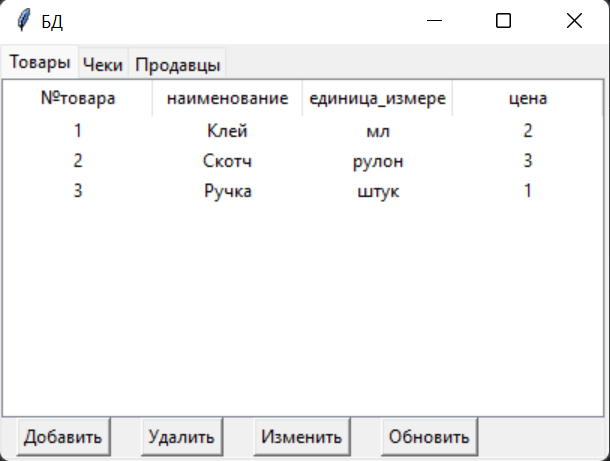
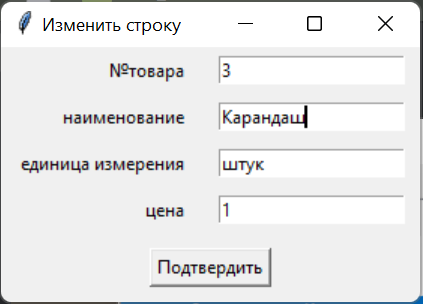




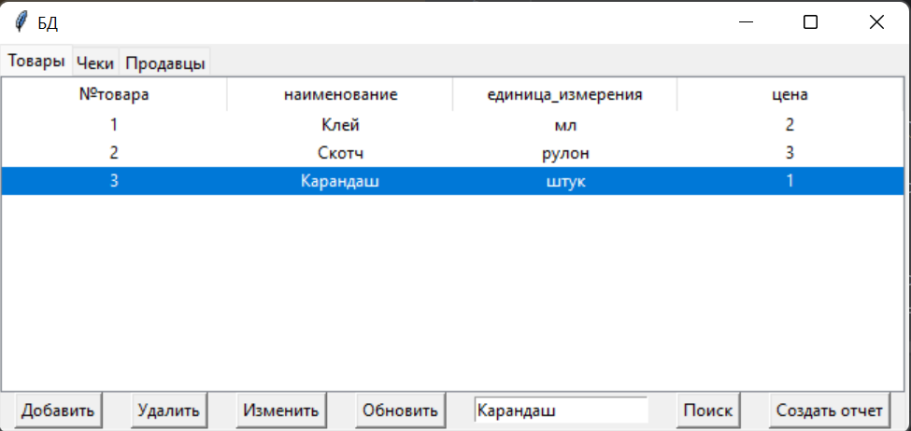
**Удаление:**



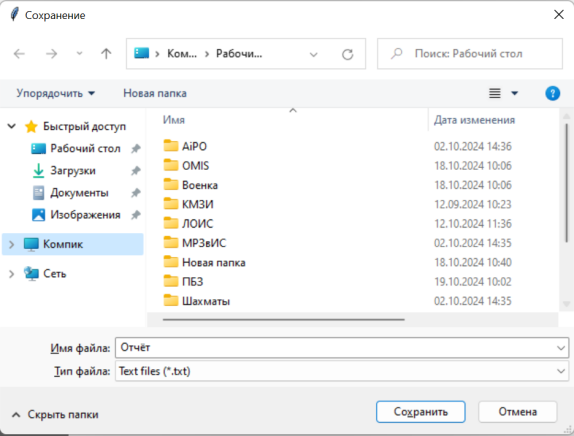
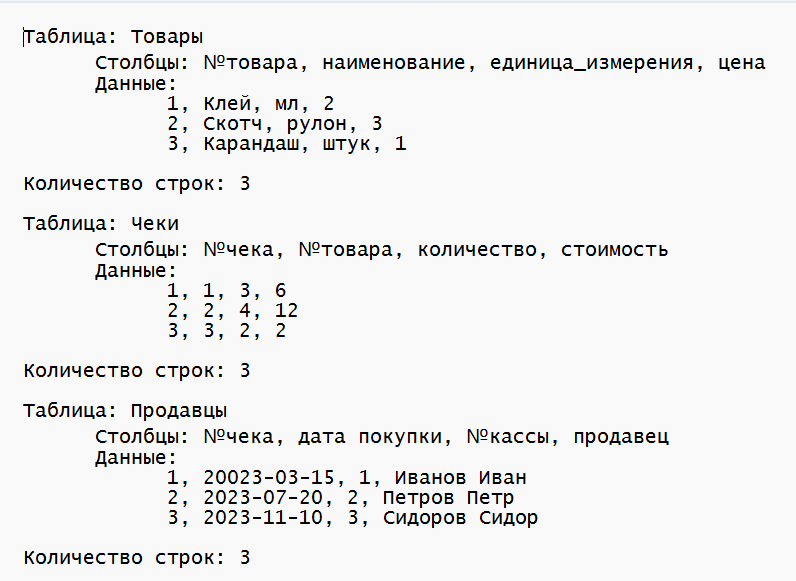
**Изменение:**



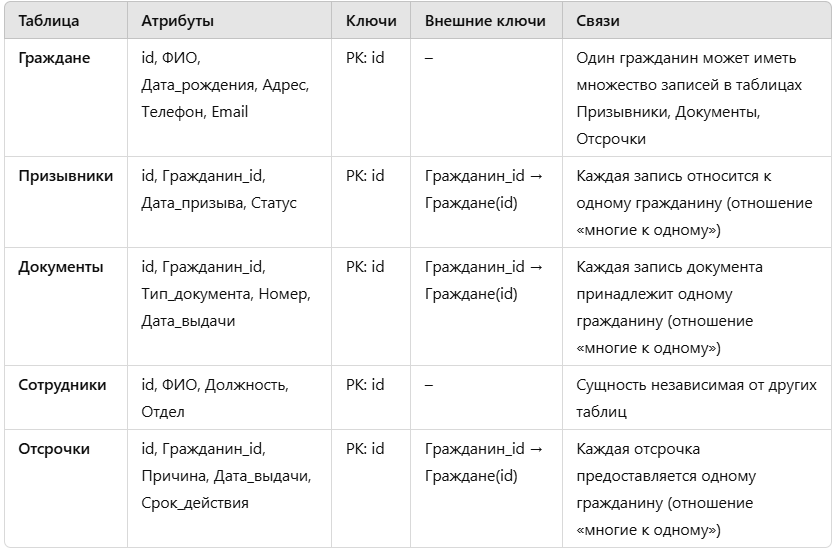
**Поиск:**



**Создание отчета:**

**Реляционная модель БД:**



**Таблица связей:**



Эта таблица показывает, что все связи идут от таблицы **Граждане** к таблицам **Призывники**, **Документы** и **Отсрочки** через внешний ключ Гражданин\_id, что соответствует отношению "один ко многим". Таблица **Сотрудники** не имеет внешних ключей и является самостоятельной сущностью.

**Вывод**: в ходе лабораторной работы я создал приложение с графическим интерфейсом для управления базой данных.