**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторная работа №6**

по дисциплине: Базы данных

тема: «Организация взаимодействия с базой данных через приложение с графическим интерфейсом»

Выполнил: ст. группы ПВ-223

Пахомов Владислав Андреевич

Проверили:

ст. пр. Панченко Максим Владимирович

Белгород 2024 г.

**Лабораторная работа №6**

Организация взаимодействия с базой данных через приложение с графическим интерфейсом  
Вариант 8

**Цель работы:** получение навыков разработки приложений для взаимодействия с базой данных, содержащих графический интерфейс пользователя.

Дополним класс Repository новыми методами. Теперь нам нужно выполнять вставку, удаление, обновление и выборку:

from tabulate import tabulate

from typing import Any

from lab6.dto.base import BaseDTOGeneartor

class Repository:

    def \_\_init\_\_(self, connection, table, generator: BaseDTOGeneartor):

        self.\_table = table

        self.generator = generator

        self.connection = connection

    def get\_dto\_generator(self):

        return self.generator

    @classmethod

    def \_get\_where\_identifier(cls, value: tuple[str, Any]) -> str:

        result = f"{value[0]}"

        if type(value[1]) is None:

            result += " IS NULL"

        elif type(value[1]) in (int, float):

            result += f"={value[1]}"

        else:

            result += f"='{value[1]}'"

        return result

    @classmethod

    def \_get\_set\_identifier(cls, value: tuple[str, Any]) -> str:

        result = f"{value[0]}="

        if type(value[1]) is None:

            result += "NULL"

        elif type(value[1]) in (int, float):

            result += f"{value[1]}"

        else:

            result += f"'{value[1]}'"

        return result

    def select(self, keys: list[str]) -> list[dict]:

        with self.connection.cursor() as cursor:

            try:

                cursor.execute(f"SELECT {','.join(keys)} FROM {self.\_table};")

                result = []

                for row in cursor.fetchall():

                    i = 0

                    new\_element = dict()

                    for key in keys:

                        new\_element[key] = row[i]

                        i += 1

                    result.append(new\_element)

                self.connection.commit()

                return result

            except Exception as e:

                self.connection.rollback()

                raise e

    def insert(self, data: dict) -> None:

        with self.connection.cursor() as cursor:

            try:

                values = map(lambda x: 'NULL' if x is None else f"'{x}'", data.values())

                cursor.execute(f"INSERT INTO {self.\_table} "

                                     f" ({', '.join(data.keys())})"

                                     f" VALUES ({', '.join(values)});")

                self.connection.commit()

            except Exception as e:

                self.connection.rollback()

                raise e

    def update(self, data: dict, identifier: dict) -> None:

        with self.connection.cursor() as cursor:

            try:

                cursor.execute(f"UPDATE {self.\_table} "

                                     f"SET {', '.join(map(Repository.\_get\_set\_identifier, data.items()))}"

                                     f"WHERE {' AND '.join(map(Repository.\_get\_where\_identifier, identifier.items()))};")

                self.connection.commit()

            except Exception as e:

                self.connection.rollback()

                raise e

    def delete(self, identifier: dict) -> None:

        with self.connection.cursor() as cursor:

            try:

                cursor.execute(f"DELETE FROM {self.\_table} "

                               f"WHERE {' AND '.join(map(Repository.\_get\_where\_identifier, identifier.items()))};")

                self.connection.commit()

            except Exception as e:

                self.connection.rollback()

                raise e

Введём понятие идентификатор – совокупность нескольких ключей, по которым можно идентифицировать запись. Она нам понадобится при удалении и обновлении. Также введём генератор датаобъектов. Датаобъекты можно представить в качестве обычных словарей. Они будут задавать структуру присылаемых данных в репозиторий. Абстрактный генератор датаклассов будет выглядеть следующим образом:

from abc import ABC, abstractmethod

class BaseDTOGeneartor(ABC):

    @abstractmethod

    def translations(self) -> dict:

        """Возвращает переводы для полей DTO"""

        pass

    @abstractmethod

    def insert(self) -> dict:

        """Возвращает базовую DTO для вставки"""

        pass

    @abstractmethod

    def select(self) -> list[str]:

        """Возвращает базовую DTO для выборки данных"""

        pass

    @abstractmethod

    def update(self) -> dict:

        """Возвращает базовую DTO для обновления данных"""

        pass

    @abstractmethod

    def identifier(self) -> dict:

        """Возвращает совокупность полей, по которым можно идентифицировать объект"""

        pass

Для отчётов будет свой репозиторий, например для отчёта неплательщиков:

from lab6.repositories.base import Repository

class NonPayersRepository(Repository):

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

        super().\_\_init\_\_(connection, "", dtogenerator)

    def select(self, keys: list[str]):

        with self.connection.cursor() as cursor:

            try:

                cursor.execute(f'''

SELECT

  resident.snp,

  SUM(payment.payment) AS debt,

  payment.energy\_source

FROM

  resident

  INNER JOIN residents\_contracts ON residents\_contracts.resident\_passport\_data = resident.passport\_data

  INNER JOIN contract ON residents\_contracts.contract\_id = contract.id

  INNER JOIN payment ON payment.contract\_id = contract.id

WHERE

  payment.paid\_date IS NULL

GROUP BY

  resident.passport\_data,

  payment.energy\_source

ORDER BY

  debt DESC;

''')

                result = []

                for row in cursor.fetchall():

                    i = 0

                    new\_element = dict()

                    for key in keys:

                        new\_element[key] = row[i]

                        i += 1

                    result.append(new\_element)

                return result

            except Exception as e:

                self.connection.rollback()

                raise e

    def insert(self, data: dict) -> None:

        raise ModuleNotFoundError("Невозможно добавить элементы в отчёт")

    def update(self, data: dict, identifier: dict) -> None:

        raise ModuleNotFoundError("Невозможно обновить элементы в отчёте")

    def delete(self, identifier: dict) -> None:

        raise ModuleNotFoundError("Невозможно удалить элемент в отчёте")

После чего сможем создать необходимые репозитории для таблиц:

from os import getenv

import psycopg2

from lab6.dto import ContractDTOGenerator, HomeDTOGenerator, PaymentDTOGenerator, ResidentDTOGenerator, \

    ResidentContractDTOGenerator, TaskDTOGenerator, WorkerDTOGenerator, WorkerTaskDTOGenerator, NonPayersDTOGenerator, \

    WorkrsRatingDTOGenerator, HomeProfitDTOGenerator

from lab6.repositories.base import Repository

from lab6.repositories.non\_payers\_repository import NonPayersRepository

from lab6.repositories.profit\_house\_repository import ProfitHouseRepository

from lab6.repositories.workers\_rating\_repository import WorkersRatingRepository

connection = psycopg2.connect(database=getenv("DATABASE"),

                              user=getenv("USERNAME"),

                              password=getenv("PASSWORD"),

                              host=getenv("HOST"),

                              port=int(getenv("PORT")),

                              options=f"-c search\_path={getenv('SCHEMA')}"

                              )

contract\_repository = Repository(connection, "contract", ContractDTOGenerator())

home\_repository = Repository(connection, "home", HomeDTOGenerator())

payment\_repository = Repository(connection, "payment", PaymentDTOGenerator())

resident\_repository = Repository(connection, "resident", ResidentDTOGenerator())

residents\_contracts\_repository = Repository(connection, "residents\_contracts", ResidentContractDTOGenerator())

task\_repository = Repository(connection, "task", TaskDTOGenerator())

worker\_repository = Repository(connection, "worker", WorkerDTOGenerator())

workers\_tasks\_repository = Repository(connection, "workers\_tasks", WorkerTaskDTOGenerator())

non\_payers\_repository = NonPayersRepository(connection, NonPayersDTOGenerator())

workers\_rating\_repository = WorkersRatingRepository(connection, WorkrsRatingDTOGenerator())

profit\_house\_repository = ProfitHouseRepository(connection, HomeProfitDTOGenerator())

all\_repos = [

    {"repo": contract\_repository, "name": "Договоры"},

    {"repo": home\_repository, "name": "Дома"},

    {"repo": payment\_repository, "name": "Чеки"},

    {"repo": resident\_repository, "name": "Жильцы"},

    {"repo": residents\_contracts\_repository, "name": "Договоры жильцов"},

    {"repo": task\_repository, "name": "Работы"},

    {"repo": worker\_repository, "name": "Исполнители работ"},

    {"repo": workers\_tasks\_repository, "name": "Назначения работ"},

    {"repo": non\_payers\_repository, "name": "Жильцы-неплательщики"},

    {"repo": workers\_rating\_repository, "name": "Рейтинг рабочих"},

    {"repo": profit\_house\_repository, "name": "Прибыль домов"}

]

Теперь мы готовы для отображения данных. В main будем инициализировать табы приложения, он довольно простой:

from PySide6.QtWidgets import QMainWindow

from lab6.repositories.repositories import all\_repos

from lab6.widgets.main\_ui import Ui\_MainWindow

from lab6.widgets.repotab import RepoTab

class MainDialog(QMainWindow):

    def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent=parent)

        self.ui = Ui\_MainWindow()

        self.ui.setupUi(self)

        self.\_\_tabs = []

        for repo\_item in all\_repos:

            tab\_test = RepoTab(repo\_item["repo"])

            self.\_\_tabs.append(tab\_test)

            self.ui.tabs.addTab(tab\_test, repo\_item["name"])

        self.ui.tabs.tabBarClicked.connect(self.\_\_upadte\_tab\_on\_select)

    def \_\_upadte\_tab\_on\_select(self, index):

        self.\_\_tabs[index].refetch\_table()

В самом RepoTab происходит взаимодействие с репозиторием. Здесь мы подгружаем данные, а также показываем необходимые диалоговые окна при выбранном действии и ошибках. При обновлении поля автоматически подгружаются в форму. Если нужно, выключаем кнопки:

import PySide6.QtCore

import PySide6.QtWidgets

from PySide6.QtWidgets import QWidget, QTableWidgetItem, QDialog

from lab6.repositories.base import Repository

from lab6.widgets.accept\_reject import AcceptRejectDialog

from lab6.widgets.form\_dialog import FormDialog

from lab6.widgets.repotab\_ui import Ui\_Form

class RepoTab(QWidget):

    def \_\_init\_\_(self, repository: Repository, parent=None) -> None:

        super().\_\_init\_\_(parent)

        self.ui = Ui\_Form()

        self.ui.setupUi(self)

        self.\_\_repository = repository

        self.\_\_values = []

        # Прячем вставку, если DTO вставки пустой

        if len(self.\_\_repository.generator.insert().values()) == 0:

            self.ui.pushButton.setVisible(False)

        # Прячем обновление, если нечего обновляьт или нельзя идентифицировать запись

        if len(self.\_\_repository.generator.identifier()) == 0 or len(self.\_\_repository.generator.update()) == 0:

            self.ui.pushButton\_2.setVisible(False)

        # Прячем удаление, если нельзя идентифицировать запись

        if len(self.\_\_repository.generator.identifier()) == 0:

            self.ui.pushButton\_3.setVisible(False)

        self.ui.pushButton\_3.clicked.connect(self.\_\_delete\_clicked)

        self.ui.pushButton\_2.clicked.connect(self.\_\_update\_clicked)

        self.ui.pushButton.clicked.connect(self.\_\_create\_clicked)

        self.refetch\_table()

    def refetch\_table(self):

        select\_keys = self.\_\_repository.generator.select()

        self.\_\_values = self.\_\_repository.select(select\_keys)

        self.\_\_redraw\_table()

    def \_\_redraw\_table(self):

        translations = self.\_\_repository.generator.translations()

        select\_keys = self.\_\_repository.generator.select()

        self.ui.tableWidget.setColumnCount(len(select\_keys))

        self.ui.tableWidget.setRowCount(len(self.\_\_values))

        self.ui.tableWidget.setHorizontalHeaderLabels(list(map(lambda x: translations[x], select\_keys)))

        i = 0

        for value in self.\_\_values:

            j = 0

            for key in select\_keys:

                self.ui.tableWidget.setItem(i, j, QTableWidgetItem("Пусто" if value[key] is None else str(value[key])))

                j += 1

            i += 1

        self.ui.tableWidget.resizeColumnsToContents()

    def \_\_create\_clicked(self):

        form\_dialog = FormDialog("Создать",

                                 list(self.\_\_repository.generator.insert().keys()),

                                 self.\_\_repository.generator.translations(),

                                 self)

        if form\_dialog.exec() == 1:

            try:

                self.\_\_repository.insert(form\_dialog.getInfo())

                self.refetch\_table()

            except Exception as e:

                whoops = AcceptRejectDialog(parent=self,

                                            title="Произошла ошибка",

                                            text=repr(e))

                whoops.show()

    def \_\_delete\_clicked(self):

        should\_delete = AcceptRejectDialog(parent=self,

                                           title="Удалить?",

                                           text=f"Вы собираетесь удалить записи ({len(self.ui.tableWidget.selectionModel().selectedRows())})")

        if should\_delete.exec() == 1:

            for row\_index in self.ui.tableWidget.selectionModel().selectedRows():

                selected\_value = self.\_\_values[row\_index.row()]

                selected\_value\_identififer = self.\_\_repository.generator.identifier()

                select\_keys = self.\_\_repository.generator.select()

                for key\_index in range(0, len(select\_keys)):

                    if select\_keys[key\_index] in selected\_value\_identififer.keys():

                        selected\_value\_identififer[select\_keys[key\_index]] = selected\_value[select\_keys[key\_index]]

                try:

                    self.\_\_repository.delete(selected\_value\_identififer)

                except Exception as e:

                    whoops = AcceptRejectDialog(parent=self,

                                                title="Произошла ошибка",

                                                text=repr(e))

                    whoops.show()

            self.refetch\_table()

    def \_\_update\_clicked(self):

        if len(self.ui.tableWidget.selectionModel().selectedRows()) == 0:

            no\_update = AcceptRejectDialog(parent=self,

                                               title="Нечего обновлять",

                                               text=f"Выберите один ряд для обновления")

            no\_update.exec()

            return

        row\_index = self.ui.tableWidget.selectionModel().selectedRows()[0]

        selected\_value = self.\_\_values[row\_index.row()]

        selected\_value\_identififer = self.\_\_repository.generator.identifier()

        select\_keys = self.\_\_repository.generator.select()

        for key\_index in range(0, len(select\_keys)):

            if select\_keys[key\_index] in selected\_value\_identififer.keys():

                selected\_value\_identififer[select\_keys[key\_index]] = selected\_value[select\_keys[key\_index]]

        form\_dialog = FormDialog("Обновить",

                                 list(self.\_\_repository.generator.update().keys()),

                                 self.\_\_repository.generator.translations(),

                                 self,

                                 selected\_value)

        if form\_dialog.exec() == 1:

            try:

                self.\_\_repository.update(form\_dialog.getInfo(), selected\_value\_identififer)

                self.refetch\_table()

            except Exception as e:

                whoops = AcceptRejectDialog(parent=self,

                                               title="Произошла ошибка",

                                               text=repr(e))

                whoops.show()

Диалоговые окна с формами и сообщением тоже относительно простые:

import datetime

from PySide6.QtWidgets import QDialog, QGroupBox, QVBoxLayout, QDialogButtonBox, QLineEdit, QFormLayout, QLabel

class FormDialog(QDialog):

    # constructor

    def \_\_init\_\_(self, title, values, translations, parent, default\_values = None):

        super(FormDialog, self).\_\_init\_\_(parent=parent)

        self.setWindowTitle(title)

        self.setGeometry(100, 100, 300, 400)

        self.formGroupBox = QGroupBox(title)

        self.formValues = []

        self.\_values = values

        self.\_translations = translations

        self.\_default\_values = default\_values

        for value in translations:

            self.formValues.append(QLineEdit())

        self.createForm()

        self.buttonBox = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)

        self.buttonBox.accepted.connect(self.accept)

        self.buttonBox.rejected.connect(self.reject)

        mainLayout = QVBoxLayout()

        mainLayout.addWidget(self.formGroupBox)

        mainLayout.addWidget(self.buttonBox)

        self.setLayout(mainLayout)

    def getInfo(self):

        answer = {}

        for i in range(0, len(self.\_values)):

            # Это просто ужасно, мы отнимаем у юзера возможность ввести None...

            answer[self.\_values[i]] = None if self.formValues[i].text() == "None" else self.formValues[i].text()

        return answer

    def createForm(self):

        layout = QFormLayout()

        for i in range(0, len(self.\_values)):

            layout.addRow(QLabel(self.\_translations[self.\_values[i]]), self.formValues[i])

            if self.\_default\_values and self.\_values[i] in self.\_default\_values:

                if type(self.\_default\_values[self.\_values[i]]) is None:

                    self.formValues[i].setText("None")

                elif type(self.\_default\_values[self.\_values[i]]) is datetime.date:

                    self.formValues[i].setText(self.\_default\_values[self.\_values[i]].isoformat())

                else:

                    self.formValues[i].setText(str(self.\_default\_values[self.\_values[i]]))

        self.formGroupBox.setLayout(layout)

from PySide6.QtWidgets import QDialog, QGroupBox, QVBoxLayout, QDialogButtonBox, QLineEdit, QFormLayout, QLabel

class AcceptRejectDialog(QDialog):

    # constructor

    def \_\_init\_\_(self, title, text, parent=None):

        super(AcceptRejectDialog, self).\_\_init\_\_(parent=parent)

        self.setWindowTitle(title)

        self.setGeometry(100, 100, 300, 200)

        self.buttonBox = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)

        self.buttonBox.accepted.connect(self.accept)

        self.buttonBox.rejected.connect(self.reject)

        mainLayout = QVBoxLayout()

        mainLayout.addWidget(QLabel(text))

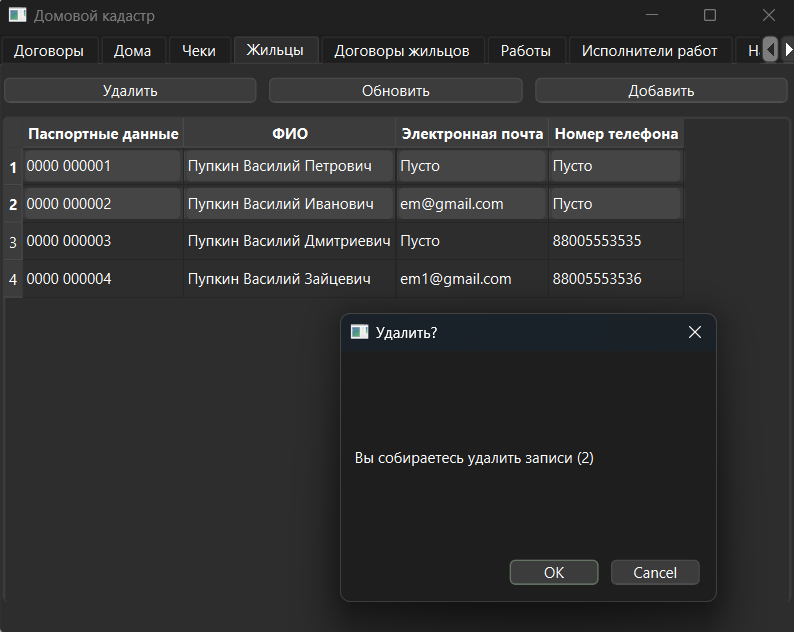
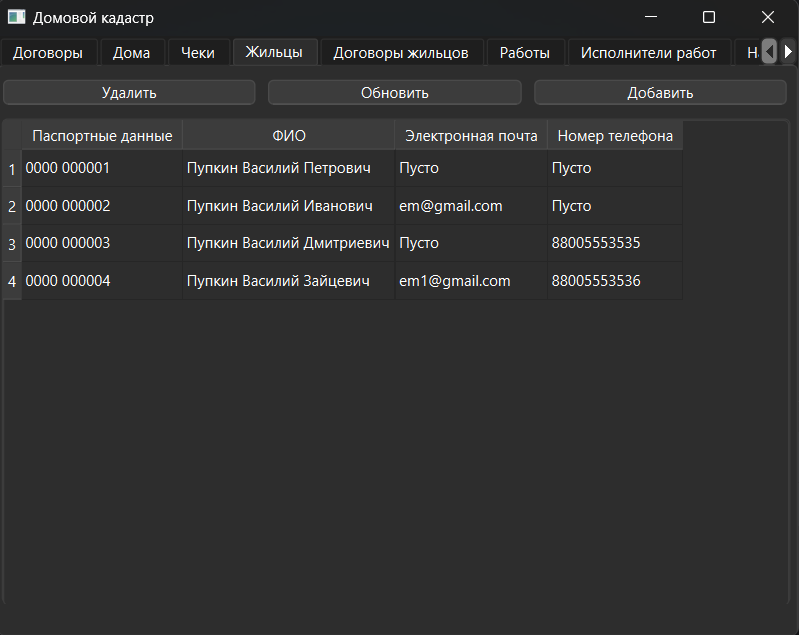
        mainLayout.addWidget(self.buttonBox)

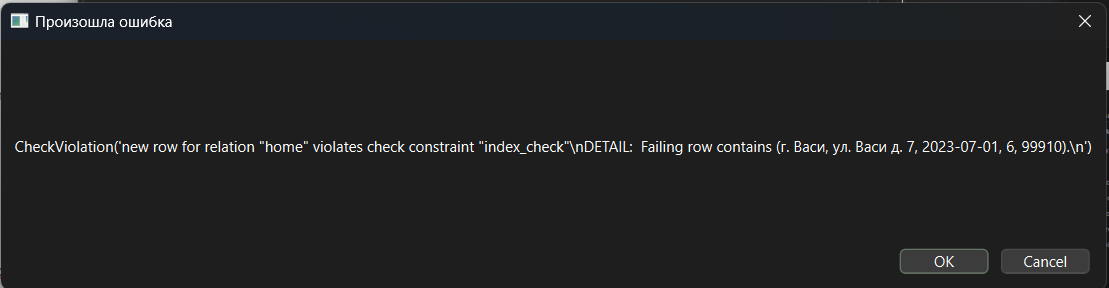
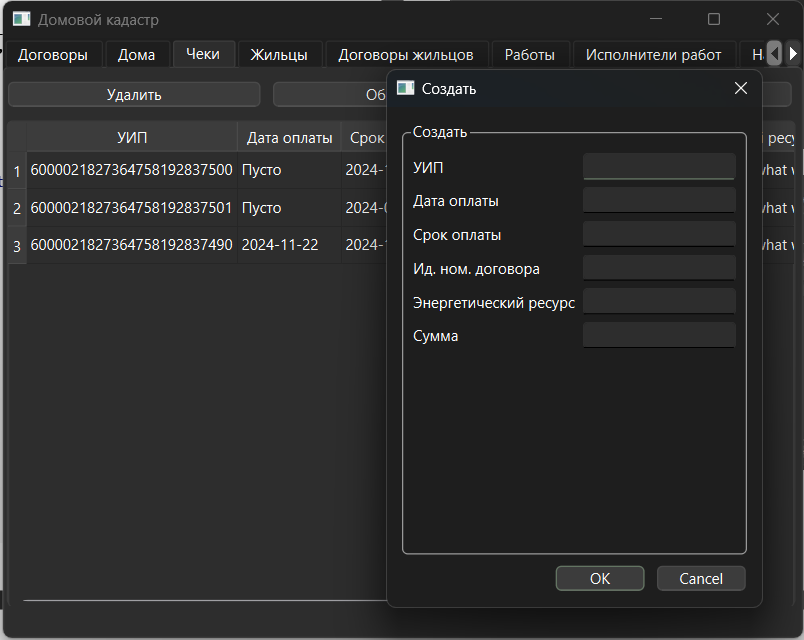
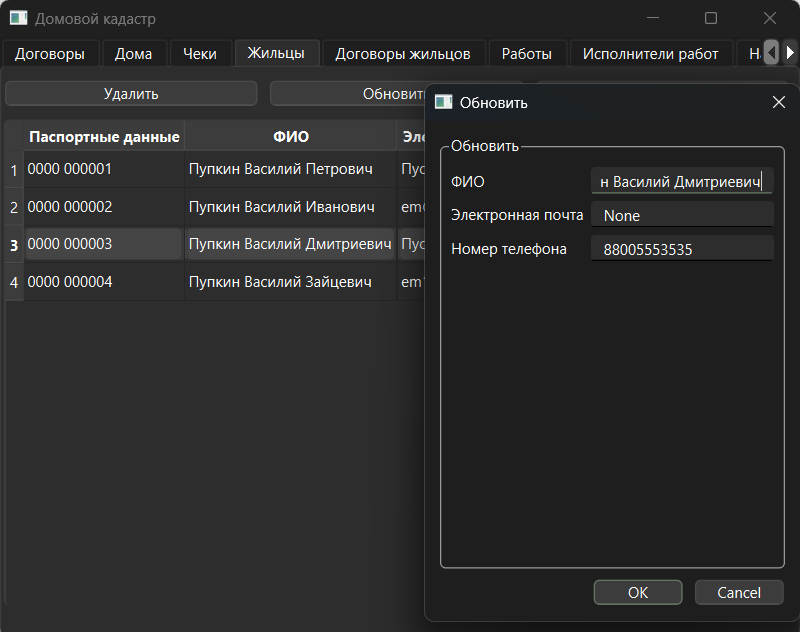
        self.setLayout(mainLayout)

Разработанное решение хорошо подходит для простых CRUD-приложений с базовыми операциями, можно относительно быстро и легко создать свой репозиторий, генератор DTO и включить Tab для отображения. Однако более сложные взаимодействия с БД приводят к написанию кастомных репозиториев и запросов, репозиторий обладает низкой степенью защищённости – никто не мешает передавать в него объект, не являющийся результатом выполнения генератора DTO. Решение обладает низкой степенью расширяемости. Нет отображения читаемых ошибок, валидация значений происходит на стороне БД, что в современных продуктах не используется. Проблему можно было бы решить при помощи предварительного проектирования архитектуры приложения, использования паттернов, однако этап проектирования занял бы неоправданно большое количество времени, кроме того, расширяемость для данного решения не понадобится в будущем, поэтому можем остановиться на текущем решении.

Ссылка на репозиторий: <https://github.com/IAmProgrammist/database/tree/main/lab6>

Результат выполнения программы:





**Вывод:** в ходе лабораторной работы получили навыки разработки приложений для взаимодействия с базой данных, содержащих графический интерфейс пользователя.