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)
```

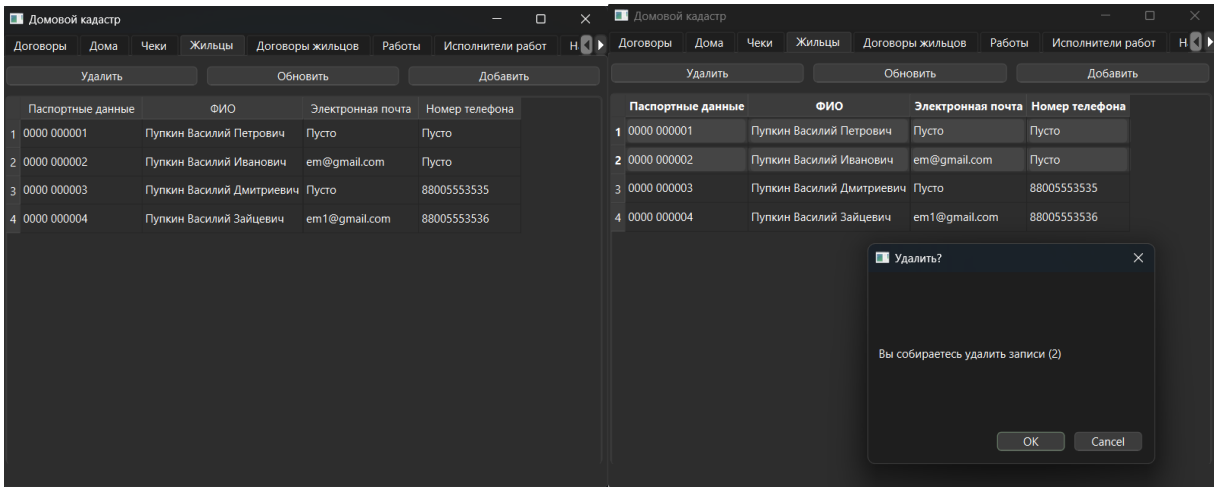
```
    def getInfo(self):
        answer = {}

        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)
```

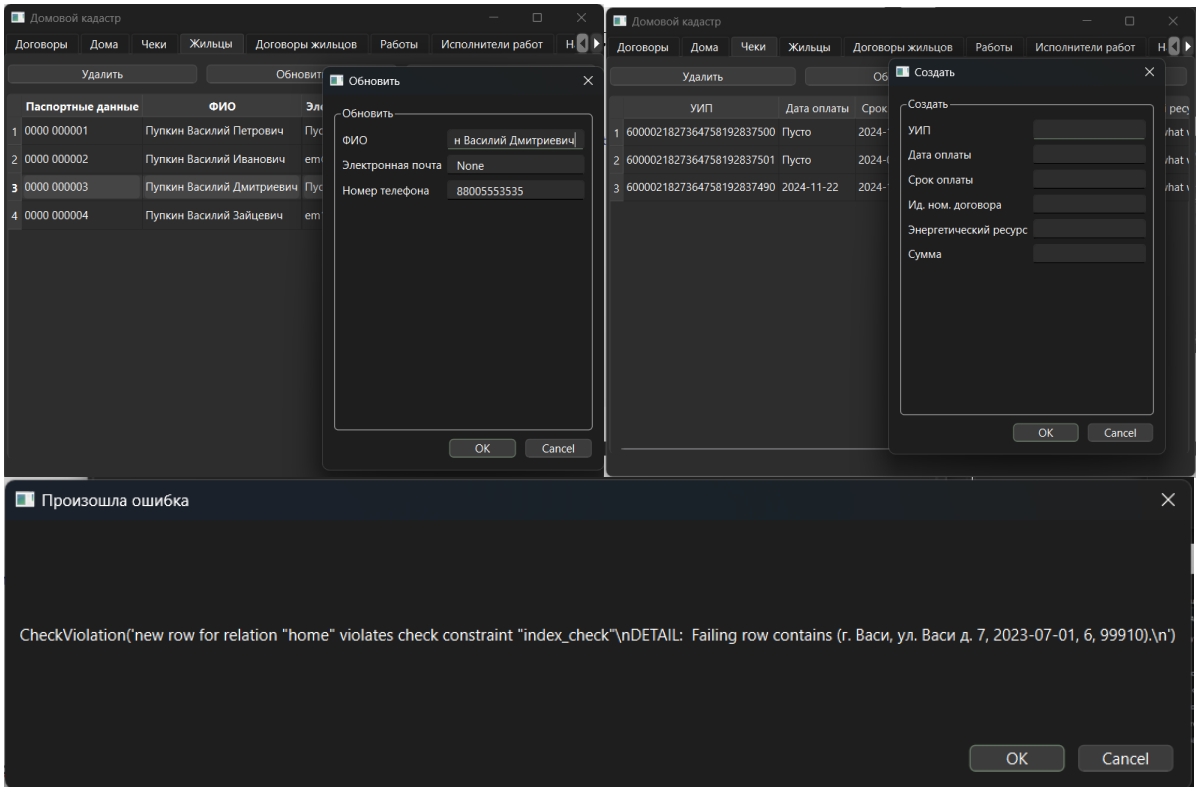
```
from PySide6.QtWidgets import QDialog, QGroupBox, QVBoxLayout, QDialogButtonBox, QLineEdit,
QFormLayout, QLabel

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.