**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 №5**

по дисциплине: Базы данных

тема: «Организация взаимодействия с базой данных через консольное приложение»

Выполнил: ст. группы ПВ-223

Пахомов Владислав Андреевич

Проверили:

ст. пр. Панченко Максим Владимирович

Белгород 2024 г.

**Лабораторная работа №5**

Организация взаимодействия с базой данных через консольное приложение  
Вариант 8

**Цель работы:** получить навыки подключения к различным системам управления базами данных и взаимодействия с ними. Разработать консольное приложение для взаимодействия с базой данных.

В качестве БД и СУБД был выбран Postgres, подключение к нему можно осуществить через библиотеку psycopg2. Для форматированного вывода использовали tabulate, для получения данных для подключения к СУБД из переменных окружения использовали getenv из os.

from tabulate import tabulate  
from os import getenv  
import psycopg2

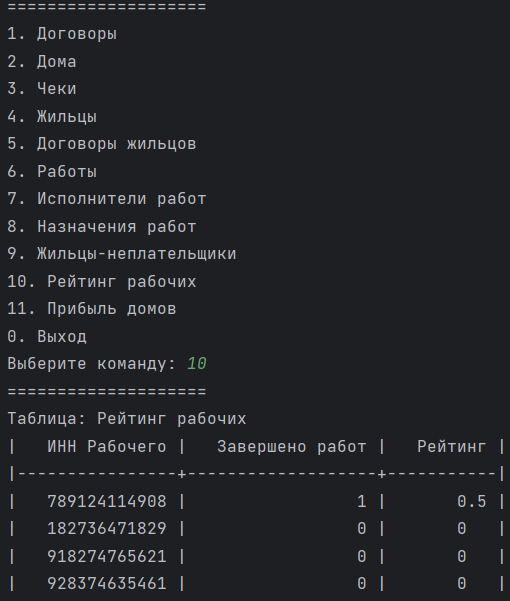
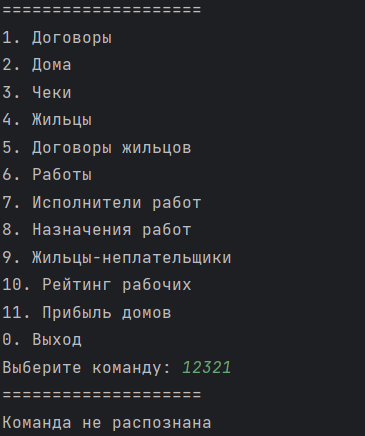
Класс Repository выполняет взаимодействие с базой данных, для получения отчётов будем использовать классы, наследующие Repository, где мы переопределяем запрос для выборки.

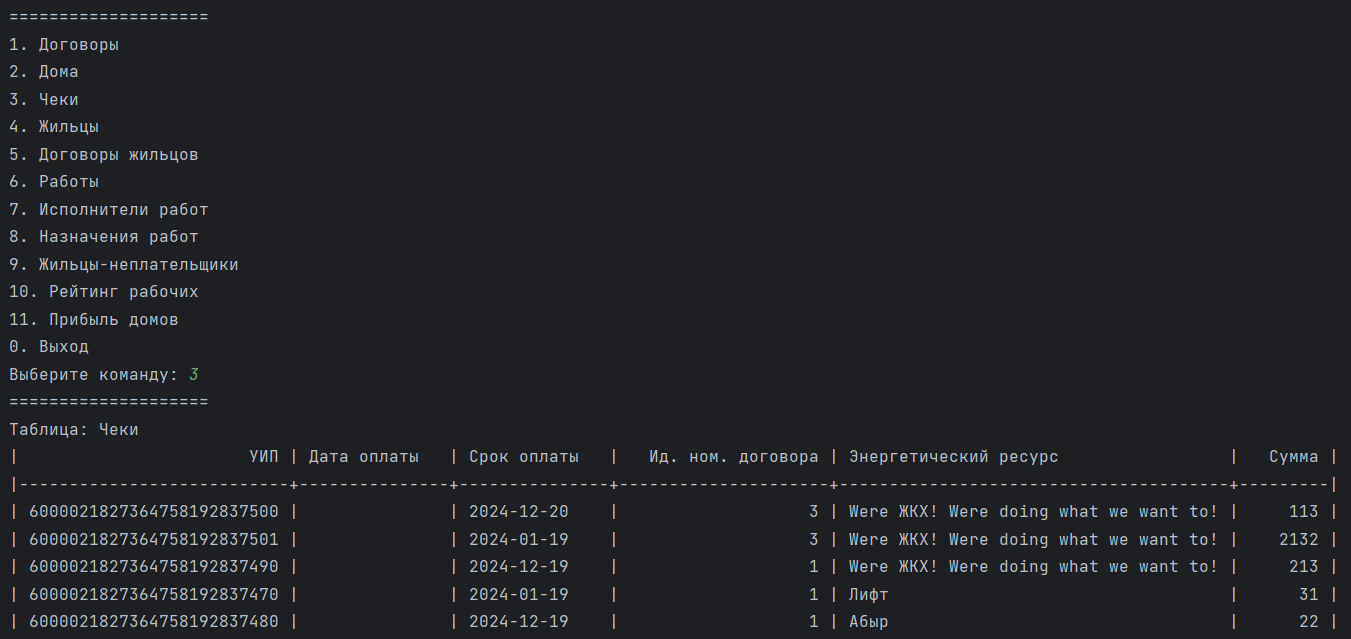
class Repository:  
 def \_\_init\_\_(self, connection, table, name, cols, cols\_names):  
 self.\_cursor = connection.cursor()  
 self.\_table = table  
 self.\_name = name  
 self.\_cols = cols  
 self.\_cols\_names = cols\_names  
  
 def select(self):  
 print(f"Таблица: {self.\_name}")  
 self.\_cursor.execute(f"SELECT {','.join(self.\_cols)} FROM {self.\_table};")  
 print(tabulate(self.\_cursor.fetchall(), headers=self.\_cols\_names, tablefmt="orgtbl"))  
  
  
class NonPayersRepository(Repository):  
 def \_\_init\_\_(self, connection, name, cols\_names):  
 super().\_\_init\_\_(connection, "", name, [], cols\_names)  
  
 def select(self):  
 print(f"Таблица: {self.\_name}")  
 self.\_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;  
''')  
 print(tabulate(self.\_cursor.fetchall(), headers=self.\_cols\_names, tablefmt="orgtbl"))  
  
  
class WorkersRatingRepository(Repository):  
 def \_\_init\_\_(self, connection, name, cols\_names):  
 super().\_\_init\_\_(connection, "", name, [], cols\_names)  
  
 def select(self):  
 print(f"Таблица: {self.\_name}")  
 self.\_cursor.execute(f'''  
SELECT   
 worker.inn AS worker\_inn,   
 COALESCE (t1.completed, 0) as completed,   
 (  
 1.0 \* COALESCE (t1.completed, 0) / t2.total  
 ) as rating   
FROM   
 worker   
 LEFT JOIN (  
 SELECT   
 workers\_tasks.worker\_inn as worker\_inn,   
 COUNT(\*) as completed   
 FROM   
 workers\_tasks   
 INNER JOIN task ON task.id = workers\_tasks.task\_id   
 WHERE   
 workers\_tasks.worker\_inn = worker\_inn   
 AND task.completed\_date IS NOT NULL   
 AND task.until\_date > '2004-01-01'  
 AND task.until\_date < '2040-12-12'  
 GROUP BY   
 workers\_tasks.worker\_inn  
 ) t1 ON t1.worker\_inn = worker.inn   
 INNER JOIN (  
 SELECT   
 workers\_tasks.worker\_inn as worker\_inn,   
 COUNT(\*) as total   
 FROM   
 workers\_tasks   
 INNER JOIN task ON task.id = workers\_tasks.task\_id   
 WHERE   
 workers\_tasks.worker\_inn = worker\_inn   
 AND task.until\_date > '2004-01-01'  
 AND task.until\_date < '2040-12-12'  
 GROUP BY   
 workers\_tasks.worker\_inn  
 ) t2 ON t2.worker\_inn = worker.inn   
ORDER BY   
 completed desc;  
''')  
 print(tabulate(self.\_cursor.fetchall(), headers=self.\_cols\_names, tablefmt="orgtbl"))  
  
  
class ProfitHousesRepository(Repository):  
 def \_\_init\_\_(self, connection, name, cols\_names):  
 super().\_\_init\_\_(connection, "", name, [], cols\_names)  
  
 def select(self):  
 print(f"Таблица: {self.\_name}")  
 self.\_cursor.execute(f'''  
SELECT   
 home.address as address,   
 (  
 COALESCE(t1.plus, 0) - COALESCE(t2.minus, 0)  
 ) as profit   
FROM   
 home   
 LEFT JOIN (  
 SELECT   
 contract.home AS home,   
 SUM(payment.payment) as plus   
 FROM   
 payment   
 LEFT JOIN contract ON contract.id = payment.contract\_id   
 GROUP BY   
 contract.home  
 ) as t1 ON t1.home = address   
 LEFT JOIN (  
 SELECT   
 task.home AS home,   
 SUM(task.payment) as minus   
 FROM   
 task   
 GROUP BY   
 task.home  
 ) as t2 ON t2.home = address   
ORDER BY   
 profit;  
''')  
 print(tabulate(self.\_cursor.fetchall(), headers=self.\_cols\_names, tablefmt="orgtbl"))

В main создадим маппинг пользовательского ввода и репозиториев, на его основе будем выводить команды и вызывать нужный репозиторий. Если репозиторий недоступен, будем сообщать об ошибке. При вводе 0 выходим из цикла:

def main():  
 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')}"  
 )  
  
 repo\_mapping = {  
 "1": ("Договоры", Repository(connection, "contract", "Договоры",  
 ["transaction\_date", "until\_date", "home", "id"],  
 ["Дата начала", "Дата окончания", "Дом", "Идентификационный номер"])),  
 "2": ("Дома", Repository(connection, "home", "Дома",  
 ["address", "commisioning", "floors", "index"],  
 ["Адрес", "Дата введения в эксплуатацию", "Этажность", "Индекс"])),  
 "3": ("Чеки", Repository(connection, "payment", "Чеки",  
 ["id", "paid\_date", "until\_date", "contract\_id", "energy\_source", "payment"],  
 ["УИП", "Дата оплаты", "Срок оплаты", "Ид. ном. договора", "Энергетический ресурс",  
 "Сумма"])),  
 "4": ("Жильцы", Repository(connection, "resident", "Жильцы",  
 ["passport\_data", "snp", "email", "phone"],  
 ["Паспортные данные", "ФИО", "Электронная почта", "Номер телефона"])),  
 "5": ("Договоры жильцов", Repository(connection, "residents\_contracts", "Договоры жильцов",  
 ["resident\_passport\_data", "contract\_id"],  
 ["Паспортные данные", "Ид. ном. договора"])),  
 "6": ("Работы", Repository(connection, "task", "Работы",  
 ["id", "completed\_date", "until\_date", "home"],  
 ["Идентификационный номер", "Дата окончания", "Дедлайн", "Дом"])),  
 "7": ("Исполнители работ", Repository(connection, "worker", "Исполнители работ",  
 ["inn", "email", "phone"],  
 ["ИНН", "Электронная почта", "Номер телефона"])),  
 "8": ("Назначения работ", Repository(connection, "workers\_tasks", "Назначения работ",  
 ["worker\_inn", "task\_id"],  
 ["ИНН исполнителя", "Ид. ном. работы"])),  
 "9": ("Жильцы-неплательщики", NonPayersRepository(connection, "Жильцы-неплательщики",  
 ["ФИО", "Долг", "Энергетический ресурс"])),  
 "10": ("Рейтинг рабочих", WorkersRatingRepository(connection, "Рейтинг рабочих",  
 ["ИНН Рабочего", "Завершено работ", "Рейтинг"])),  
 "11": ("Прибыль домов", ProfitHousesRepository(connection, "Прибыль домов",  
 ["Адрес", "Прибыль"])),  
  
 }  
  
 repo\_selects = dict()  
 for key, val in repo\_mapping.items():  
 repo\_selects[key] = val[0]  
  
 repo\_selects["0"] = "Выход"  
  
 print("=" \* 20)  
 for key, val in repo\_selects.items():  
 print(key, val, sep=". ")  
  
 input\_value = input("Выберите команду: ")  
 print("=" \* 20)  
  
 while input\_value != "0":  
 if input\_value in repo\_mapping:  
 repo\_mapping[input\_value][1].select()  
 else:  
 print("Команда не распознана")  
  
 print("=" \* 20)  
 for key, val in repo\_selects.items():  
 print(key, val, sep=". ")  
  
 input\_value = input("Выберите команду: ")  
 print("=" \* 20)

Результаты выполнения программы:





**Вывод:** в ходе лабораторной работы получили навыки подключения к различным системам управления базами данных и взаимодействия с ними. Разработали консольное приложение для взаимодействия с базой данных.