РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №5

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

Выполнил: ст. группы ПВ-201
Барышникова Варвара Дмитриевна

Проверил:
Кулешова Екатерина Анатольевна

Белгород 2022 г.

Лабораторная работа №5

Организация взаимодействия с базой данных через консольное приложение

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

Задание к работе

1. Подключиться к системе управления базами данных с помощью SQL библиотек выбранного языка программирования.

2. Организовать взаимодействие с базами данных выбранной СУБД (PostgreSQL/MySQL/SQLite).

3. Разработать консольное приложение, в котором производится подключение к базе данных, разработанной на основе предыдущих лабораторных работ, а также обеспечивается выполнение запросов предшествующей лабораторной работы.

Выполнение работы:

Код работы:

```
import psycopg2
from typing import List, Optional, Tuple, Callable
from prettytable import PrettyTable


class ServiceBD:
    def __init__(self, name_db: str):
        self.__connection = psycopg2.connect(database=name_db,
                                              user="postgres",
                                              password="134340",
                                              host="localhost",
                                              port=5432)
        self.__cursor = self.__connection.cursor()
        self.__name_db = name_db

    def select(
            self,
            table: str,
            joins: Optional[list[tuple[str, str, str]]] = None,
            fields: Optional[list[str]] = None,
            group_by: Optional[list[str]] = None,
            order_by: Optional[str] = None,
            where: Optional[list[str]] = None
    ):
        fields = ['*'] if fields is None else fields
        query = f'SET search_path TO {self.__name_db}; '
        query += f'SELECT {", ".join(fields)} FROM {table}'
        if joins:
            query += ' JOIN ' + ' JOIN '.join([f'{tab} AS {short} ON {rule}' for tab,
short, rule in joins])
        if where:
```

```
            query += ' WHERE ' + ' AND '.join([f'{condition}' for condition in
where])
        if group_by:
            query += ' GROUP BY ' + ','.join([f'{group} ' for group in group_by])
        if order_by and not order_by.startswith('-'):
            query += f' ORDER BY {order_by} DESC'
        elif order_by:
            query += f' ORDER BY {order_by[1:]} ASC'
        query += ';'

        self.__cursor.execute(query)
        result = self.__cursor.fetchall()

        return result

    def execute_select(self, select_str: str):
        self.__cursor.execute(select_str)
        result = self.__cursor.fetchall()

        return result


class LenMenu:
    def __init__(self):
        self.__db = ServiceBD("len_ceramic")
        self.__entities = {
            1: 'Таблицы',
            2: 'Запросы',
            0: 'Выход',
        }

        self.__tables = {
            1: 'Клиенты',
            2: 'Статусы клиентов',
            3: 'Типы клиентов',
            4: 'Группы клиентов',
            5: 'Контактные данные',
            6: 'Типы контактных данных',
            7: 'Контактные данные и клиенты',
            8: 'Соц сети',
            9: 'Соц сети контактных данных',
            10: 'Фидбек',
            11: 'Сообщения',
            12: 'Шаблоны сообщений',
            13: 'Занятия',
            14: 'Типы занятий',
            15: 'Гости',
            16: 'Изделия',
            17: 'Типы фото',
            18: 'Типы изделий',
            19: 'Состояния готовности',
            20: 'Фото изделий',
            21: 'Типы активностей',
            22: 'Активности',
            23: 'Счёты',
            24: 'Сертификаты',
            25: 'Типы оплаты',
            26: 'Платежи',
        }

        self.__queries = {
            1: 'Выбрать все изделия, оплаченные сертификатом',
```

```
            2: 'Выбрать гостей, лепивших в гончарке в августе',
            3: 'Выбрать рейтинг посетителей (топ 5) гончарки на основании любой
активности и вывести по ним сумму, '
               'заплаченную, за последний календарный год',
            4: 'Вывести для каждого клиента суммы счетов за изделия по каждому
статусу готовности',
        }

        self.__menu_tables = {
            1: (lambda:
                self.__db.select('clients'), ['id', 'name', 'fcs', 'contacts_id',
'client_type_id', 'client_status_id']
                ),

            2: (lambda:
                self.__db.select('client_statuses'), ['id', 'name', 'definition']
                ),

            3: (lambda:
                self.__db.select('client_types'), ['id', 'name', 'definition',
'start_cost']
                ),

            4: (lambda:
                self.__db.select('clients_group'), ['client_group_id', 'client_id',
'is_leader']
                ),

            5: (lambda:
                self.__db.select('contacts'),
                ['id', 'client_id', 'first_name', 'last_name', 'middle_name',
'phone', 'birthday', 'ban_on_spam']
                ),

            6: (lambda:
                self.__db.select('contacts_types'), ['id', 'name']
                ),

            7: (lambda:
                self.__db.select('clients_contacts'), ['contact_id', 'client_id',
'contacts_type_id']
                ),

            8: (lambda:
                self.__db.select('media'), ['id', 'name', 'site', 'is_legal',
'use_phone']
                ),

            9: (lambda:
                self.__db.select('contacts_media'), ['id', 'contacts_id', 'media_id',
'nickname', 'is_main']
                ),

            10: (lambda:
                 self.__db.select('feedback'), ['id', 'name']
                 ),

            11: (lambda:
                 self.__db.select('messages'),
                 ['id', 'theme', 'text', 'date_sent', 'contacts_media_id',
'feedback_id', 'template_id']
                 ),
```

```
            12: (lambda:
                 self.__db.select('message_templates'), ['id', 'theme',
'template_str']
                 ),

            13: (lambda:
                 self.__db.select('classes'), ['id', 'class_type_id', 'date', 'time',
'next_date', 'comment']
                 ),

            14: (lambda:
                 self.__db.select('class_types'), ['id', 'name', 'definition',
'start_cost']
                 ),

            15: (lambda:
                 self.__db.select('guests'), ['id', 'client_id', 'class_id',
'contacts_id']
                 ),

            16: (lambda:
                 self.__db.select('products'),
                 ['id', 'name', 'product_type_id', 'first_class_id',
'ready_state_id', 'definition', 'cost']
                 ),

            17: (lambda:
                 self.__db.select('photo_types'), ['id', 'name']
                 ),

            18: (lambda:
                 self.__db.select('product_types'), ['id', 'name', 'start_cost',
'definition']
                 ),

            19: (lambda:
                 self.__db.select('ready_states'), ['id', 'name', 'comment']
                 ),

            20: (lambda:
                 self.__db.select('product_photos'), ['id', 'class_id', 'product_id',
'photo_path', 'photo_type_id']
                 ),

            21: (lambda:
                 self.__db.select('activity_types'), ['id', 'name']
                 ),

            22: (lambda:
                 self.__db.select('activities'),
                 ['id', 'name', 'client_id', 'activity_type_id', 'comment',
'product_id']
                 ),

            23: (lambda:
                 self.__db.select('cheques'), ['id', 'client_id', 'product_id',
'class_id', 'name', 'sum', 'comment']
                 ),

            24: (lambda:
                 self.__db.select('certificates'),
```

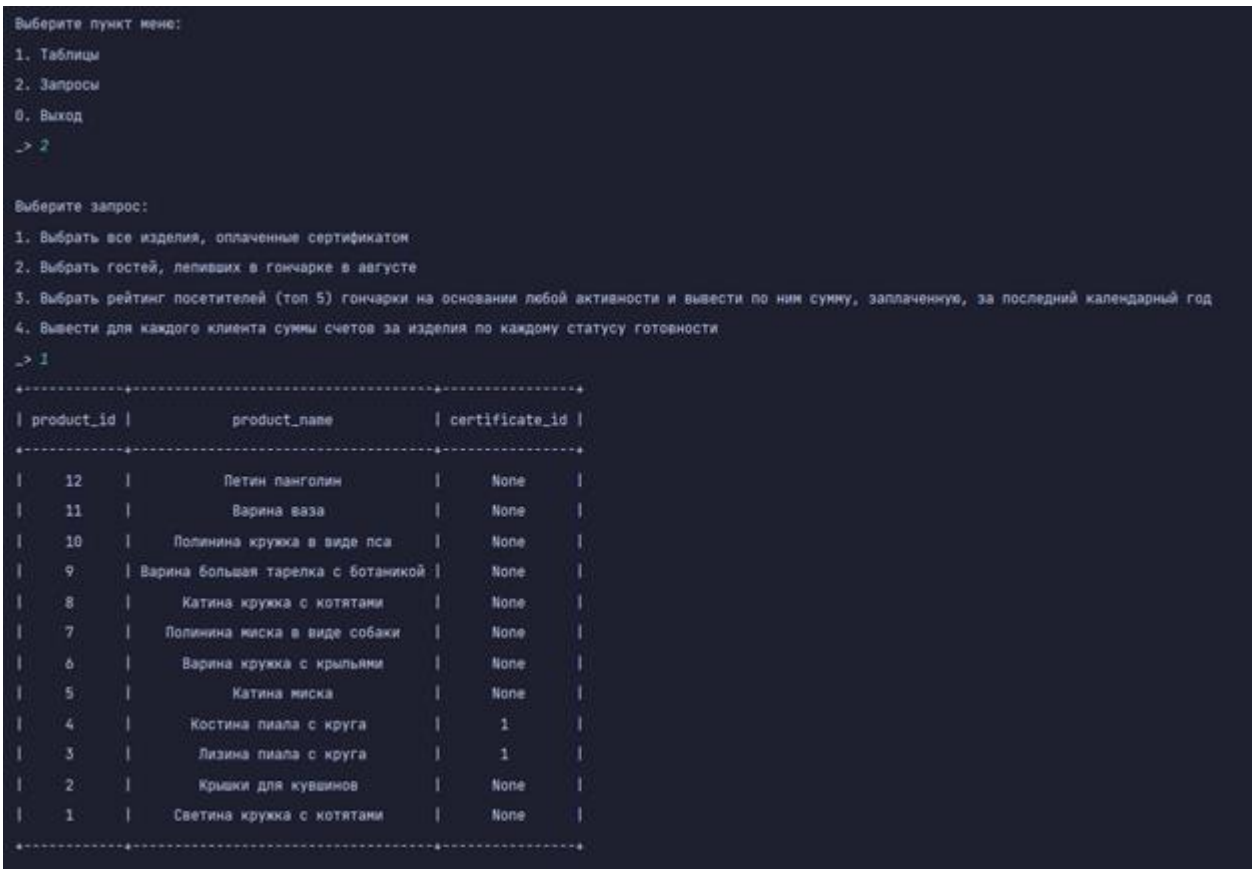
```
                ['id', 'client_buyer_id', 'client_recipient_id', 'purchase_date',
'receiving_date', 'cost',
                 'cost_used',
                 'class_type_id', 'comment']
                ),

            25: (lambda:
                 self.__db.select('paid_type'), ['id', 'name']
                 ),

            26: (lambda:
                 self.__db.select('payments'), ['id', 'cheque_id', 'sum_paid',
'paid_type_id', 'certificate_id', 'date']
                 )

        }

        self.__menu_queries = {
            1: (lambda:
                self.__db.select('products',
                                 fields=['products.id as "product_id"',
                                         'products.name as "product_name"',
                                         'payments.certificate_id as
"certificate_id"'],
                                 joins=[('cheques', 'cheques', 'products.id =
cheques.product_id '),
                                        ('payments', 'payments', 'payments.cheque_id
= cheques.id '),
                                        ('paid_type', 'paid_type',
'payments.paid_type_id = paid_type.id ')],
                                 group_by=['products.id', 'payments.certificate_id'],
                                 order_by='products.id',
                                 where=['(paid_type.name LIKE \'Сертификат%\')'],
                                 ), ['product_id', 'product_name',
'certificate_id']),

            2: (lambda:
                self.__db.execute_select(f'SELECT clients.id as "client_id", '
                                         'clients.name as "guest_name"'
                                         'FROM guests '
                                         'JOIN clients on clients.id =
guests.client_id '
                                         'JOIN classes on guests.class_id =
classes.id '
                                         'WHERE (classes.date BETWEEN \'2022-07-
31\'::date AND \'2022-09-01\'::date) '

                                         'GROUP BY clients.id, clients.name '

                                         'ORDER BY clients.id'
                                         ';'
                                         ), ['client_id', 'guest_name']),

            3: (lambda:
                self.__db.execute_select('SELECT *'
                                         'FROM (SELECT client_id,'
                                         'client_name,'
                                         'SUM(num_of_activities) as
"num_of_activities"'
                                         'FROM (SELECT clients.id           as
"client_id",'
                                         'clients.name         as "client_name",'
```

```
                                        'COUNT(DISTINCT g.id) as
"num_of_activities"'
                                        'FROM clients '
                                        'JOIN guests g on clients.id = g.client_id '
                                        'GROUP BY clients.id '
                                        'UNION ALL '
                                        'SELECT cl.id, '
                                        'cl.name, '
                                        'COUNT(DISTINCT cert.id)'
                                        'FROM certificates cert'
                                        ' JOIN clients cl on cert.client_buyer_id =
cl.id'
                                        ' GROUP BY cl.id'
                                        ' UNION ALL'
                                        ' SELECT cl.id,'
                                        ' cl.name,'
                                        ' COUNT(DISTINCT act.id)'
                                        'FROM activities act'
                                        ' JOIN clients cl on act.client_id = cl.id'
                                        ' GROUP BY cl.id) AS clients_dif_activities'
                                        ' GROUP BY client_id, client_name'
                                        ' ORDER BY num_of_activities DESC'
                                        ' LIMIT 5) AS top_activity'

                                        ' LEFT JOIN (SELECT ch.client_id as
"client_who_paid_id",'
                                        ' SUM(p.sum_paid)'
                                        'FROM cheques ch'
                                        ' JOIN payments p on ch.id = p.cheque_id'
                                        ' WHERE (p.date BETWEEN \'2021-12-31\'::date
AND \'2023-01-01\'::date)'
                                        'GROUP BY ch.client_id) AS payment ON
payment.client_who_paid_id = '
                                        'top_activity.client_id '
                                        ),
                ['client_id', 'client_name', 'num_of_classes', 'num_of_purch_cert',
'num_of_activ']
                ),

            4: (lambda:
                self.__db.execute_select(
                    'SELECT clients.id as "client_id",'
                    'clients.name as "client_name",'
                    'SUM(case when clients.id = c.client_id and rs.id = 4 then c.sum
end) as "изделие готово",'
                    'SUM(case when clients.id = c.client_id and rs.id = 3 then c.sum
end) as "ждёт второй обжиг", '
                    'SUM(case when clients.id = c.client_id and rs.id = 2 then c.sum
end) as "готово к покраске", '
                    'SUM(case when clients.id = c.client_id and rs.id = 1 then c.sum
end) as "ещё сохнет"'
                    ''
                    'FROM clients'
                    ' JOIN cheques c on clients.id = c.client_id'
                    ' JOIN products p on c.product_id = p.id'
                    ' JOIN ready_states rs on p.ready_state_id = rs.id'
                    ' GROUP BY clients.id'
                    ' ORDER BY clients.id'
                ),
                ['client_id', 'client_name', 'изделие готово', 'ждёт второй обжиг',
'готово к покраске', 'ещё сохнет']
                )
```

```
        }

    def __choose_menu(self) -> int:
        text = f'\n\nВыберите пункт меню:\n'
        for number, name in self.__entities.items():
            text += f'{number}. {name}\n'
        text += '_> '
        return int(input(text))

    def __choose_table(self) -> int:
        text = f'\nВыберите таблицу для вывода:\n'
        for number, name in self.__tables.items():
            text += f'{number}. {name}\n'
        text += '_> '
        return int(input(text))

    def __choose_queries(self) -> int:
        text = f'\nВыберите запрос:\n'
        for number, name in self.__queries.items():
            text += f'{number}. {name}\n'
        text += '_> '
        return int(input(text))

    @staticmethod
    def __show(get_rows_func, columns: list[str]):
        table = PrettyTable()
        table.field_names = columns
        for row in get_rows_func():
            table.add_row(row)
        print(table)

    def main(self):
        while True:
            command = self.__choose_menu()

            if command == 0:
                return
            elif command not in self.__entities.keys():
                raise IndexError(f'\nКоманда {command} недопустима:
{list(self.__entities.keys())}')

            if command == 1:
                command = self.__choose_table()
                if command not in self.__tables.keys():
                    raise IndexError(f'\nКоманда {command} недопустима:
{list(self.__tables.keys())}')
                get_rows_func, name_columns = self.__menu_tables[command]

            elif command == 2:
                command = self.__choose_queries()
                if command not in self.__queries.keys():
                    raise IndexError(f'\nКоманда {command} недопустима:
{list(self.__queries.keys())}')
                get_rows_func, name_columns = self.__menu_queries[command]

            self.__show(get_rows_func, name_columns)


if __name__ == "__main__":
    menu = LenMenu()
    menu.main()
```

Скриншоты работы программы:

```
Выберите пункт меню:
1. Таблицы
2. Запросы
0. Выход
_> 1

Выберите таблицу для вывода:
1. Клиенты
2. Статусы клиентов
3. Типы клиентов
4. Группы клиентов
5. Контактные данные
6. Типы контактных данных
7. Контактные данные и клиенты
8. Соц сети
9. Соц сети контактных данных
10. Фидбек
11. Сообщения
12. Шаблоны сообщений
13. Занятия
14. Типы занятий
15. Гости
16. Изделия
17. Типы фото
18. Типы изделий
19. Состояния готовности
20. Фото изделий
21. Типы активностей
22. Активности
23. Счёты
24. Сертификаты
25. Типы оплаты
26. Платежи
_> 22
+----+---------------------------+-----------+-----------------+---------+------------+
| id |           name            | client_id | activity_type_id | comment | product_id |
+----+---------------------------+-----------+-----------------+---------+------------+
| 1  | Заказ крышек для кувшинов |     6     |        1        |   None  |     2      |
+----+---------------------------+-----------+-----------------+---------+------------+
```



Вывод: в ходе выполнения данной лабораторной работы я получила навыки подключения к различным СУБД и взаимодействия с ними. Мною было разработано консольное приложение для взаимодействия с БД.