Кольца Илья Вячеславович

Лабораторная работа № 1

Вариант 3

Файл Main

```
import sys
sys.path.append('tables')

from datetime import datetime
from project_config import *
from dbconnection import *

from people_table import *
from phones_table import *
from documents_table import *


class Main:

    people_menu = """Дальнейшие операции:
    0 - возврат в главное меню;
    3 - добавление нового человека;
    4 - удаление человека;
    5 - просмотр телефонов человека;
    6 - просмотр документов человека;
    7 - постраничный вывод людей;
    9 - выход."""

    phones_menu = """Дальнейшие операции:
    0 - возврат в главное меню;
    1 - возврат в просмотр людей;
    6 - добавление нового телефона;
    7 - удаление телефона;
    9 - выход."""

    documents_menu = """Дальнейшие операции:
    0 - возврат в главное меню;
    1 - возврат в просмотр людей;
    6 - добавление нового документа;
    7 - удаление документа;
    9 - выход."""

    main_menu = """Добро пожаловать!
    Основное меню (выберите цифру в соответствии с необходимым действием):
    1 - просмотр людей;
    2 - сброс и инициализация таблиц;
    9 - выход."""

    config = ProjectConfig()
```

```
    connection = DbConnection(config)

    def __init__(self):
        DbTable.dbconn = self.connection


    def db_init(self):
        pt = PeopleTable()
        pht = PhonesTable()
        dt =  DocumentsTable()

        pt.create()
        pht.create()
        dt.create()


    def db_insert_somethings(self):
        pt = PeopleTable()
        pht = PhonesTable()
        dt = DocumentsTable()

        pt.insert_one(["Test", "Test", "Test"])
        pt.insert_one(["Test2", "Test2", "Test2"])
        pt.insert_one(["Test3", "Test3", "Test3"])

        pht.insert_one([1, "123"])
        pht.insert_one([2, "123"])
        pht.insert_one([3, "123"])
        dt.insert_one(['12.12.2020', '123456', 1, 'ГУ МВД России по городу
Москва',  '1234', 'passport'])


    def main_cycle(self):
        current_menu = "0"
        next_step = None
        while(current_menu != "9"):
            if current_menu == "0":
                self.show_main_menu()
                next_step = self.read_next_step()
                current_menu = self.after_main_menu(next_step)

            elif current_menu == "1":
                self.show_people()
                next_step = self.read_next_step()
                current_menu = self.after_show_people(next_step)

            elif current_menu == "2":
                self.show_main_menu()

            elif current_menu == "3":
                self.show_add_person()
```

```
                current_menu = "1"

        print("До свидания!")


    def db_drop(self):
        pht = PhonesTable()
        pt = PeopleTable()
        dt = DocumentsTable()
        dt.drop()
        pht.drop()
        pt.drop()


    def show_main_menu(self):
        print(self.main_menu)


    def read_next_step(self):
        return input("=> ").strip()


    def after_main_menu(self, next_step):
        if next_step == "2":
            self.db_drop()
            self.db_init()
            self.db_insert_somethings()
            print("Таблицы созданы заново!")
            return "0"
        elif next_step != "1" and next_step != "9":
            print("Выбрано неверное число! Повторите ввод!")
            return "0"
        else:
            return next_step


    def delete_by_field(self, table, menu, field):
        while True:
            num = input("Для удаления записи введите id человека, данные которого
нужно удалить (0 - отмена): ").strip()
            if self.check_sql_injections(num):
                print("Недопустимый ввод!")
                return "1"
            while len(num) == 0:
                num = input("Введена пустая строка!").strip()
                if self.check_sql_injections(num):
                    print("Недопустимый ввод!")
                    return "1"
            if not num.isnumeric():
                print("Недопустимый ввод!")
                return '1'
```

```
            if num == "0":
                return "1"
            obj = table().check_field_exist(field, num)
            if not obj:
                print(f"Неверный ввод!")
            else:
                break
        success = table().delete_one_by_field(field, num)
        print("Запись была уничтожена!" if success else "Неверный ввод")
        print(menu)
        return self.read_next_step()


    def show_people(self):
        self.person_id = -1
        print( """Просмотр списка людей!\n№\t\tФамилия\t\tИмя\t\tОтчество""")
        lst = PeopleTable().all()
        for i in lst:
            print(str(i[0]) + "\t\t" + str(i[2]) + "\t\t" + str(i[1]) + "\t\t" +
str(i[3]))
        print(self.people_menu)
        return '1'


    def after_show_people(self, next_step):
         while True:
            if next_step == "4":
                next_step = self.delete_man_by_id()

            elif next_step == "5":
                next_step = self.show_phones_by_people()
                while next_step == "6" or next_step == "7":
                    if next_step == "6":
                        next_step = self.show_add_phone()
                    elif next_step == "7":
                        next_step = self.delete_phone()
                if next_step != "0" and next_step != "1" and next_step != "9":
                    print("Выбрано неверное число! Повторите ввод!")
                    next_step = "5"

            elif next_step == "6":
                next_step = self.show_documents_by_people()
                while next_step == "6" or next_step == "7":
                    if next_step == "6":
                        next_step = self.show_add_document()
                    elif next_step == "7":
                        next_step = self.delete_document()
                if next_step != "0" and next_step != "1" and next_step != "9":
                    print("Выбрано неверное число! Повторите ввод!")
                    next_step = "6"
```

```
            elif next_step == "7":
                self.show_people_by_page()
                next_step = '1'

            elif next_step != "0" and next_step != "1" and next_step != "9" and
next_step != "3":
                print("Выбрано неверное число! Повторите ввод!")
                return "1"
            else:
                return next_step


    def input_check(self, field, length=32):
        row = input(f"Введите поле {field} (1 - отмена): ").strip()
        if self.check_sql_injections(row):
            print("Недопустимый ввод!")
            return
        while True:
            if len(row) == 0:
                if field == 'отчество':
                    return '-'
                row = input(f"Поле {field} не может быть пустым! Введите заново
(1 - отмена):").strip()
                if self.check_sql_injections(row):
                    print("Недопустимый ввод!")
                    return
            if row == "1":
                return
            if len(row) > length:
                row = input(f"Поле {field} не может быть более 32 символов.
Введите заново (1 - отмена):").strip()
                if self.check_sql_injections(row):
                    print("Недопустимый ввод!")
                    return
            else:
                return row


    def check_sql_injections(self, t):
        t = t.split()
        if 'select' in t or 'union' in t or 'order by' in t or '=' in t or 'drop'
in t or 'delete' in t or 'sleep' \
                in t or 'update' in t or 'alter' in t or 'modify' in t or ';' in
t or 'create' in t:
            return True
        else:
            return False


    def show_add_person(self):
        data = []
```

```
        for field in ['фамилия', 'имя','отчество']:
            row = self.input_check(field)
            if row is None:
                return '1'
            else:
                data.append(row)
        PeopleTable().insert_one(data)
        return '1'


    def delete_man_by_id(self):
        return self.delete_by_field(PeopleTable, self.people_menu, 'id')


    def show_people_by_page(self):
        PeopleTable().pst_read()


    def show_phones_by_people(self):
        if self.person_id == -1:
            while True:
                num = input("Укажите номер строки, в которой записана
интересующая Вас персона (0 - отмена): ")
                if self.check_sql_injections(num):
                    print("Недопустимый ввод!")
                    return
                while len(num.strip()) == 0:
                    num = input("Пустая строка. Повторите ввод! Укажите номер
строки, в которой записана интересующая Вас персона (0 - отмена): ")
                if self.check_sql_injections(num):
                    print("Недопустимый ввод!")
                    return '1'
                if num == "0":
                    return "1"
                person = PeopleTable().find_by_position(int(num))
                if not person:
                    print("Введено число, неудовлетворяющее количеству людей!")
                else:
                    self.person_id = int(person[0])
                    self.person_obj = person
                    break
        print("Выбран человек: " + self.person_obj[1] + " " + self.person_obj[2]
+ " " + self.person_obj[3])
        print("Телефоны:")
        lst = PhonesTable().all_by_person_id(self.person_id)
        for i in lst:
            print(i[1])
        print(self.phones_menu)
        return self.read_next_step()
```

```
    def show_add_phone(self):
        while True:
            phone = self.input_check('номер телефона', 12)
            if phone is None:
                return '1'
            if not phone.isnumeric():
                print('Некорректный ввод!')
            break

        exist = PhonesTable().check_field_exist(
            ['person_id', 'phone'], [self.person_id, phone])
        print(self.phones_menu)
        if exist:
            print('Нарушен Primary Key, запись не вставлена')
            return self.read_next_step()
        PhonesTable().insert_one([self.person_id, phone])
        return self.read_next_step()


    def delete_phone(self):
        return self.delete_by_field(PhonesTable, self.phones_menu, 'person_id')


    def show_documents_by_people(self):
        if self.person_id == -1:
            while True:
                num = input("Введите id человека (0 - отмена):")
                if self.check_sql_injections(num):
                    print("Недопустимый ввод!")
                    return '1'
                while len(num.strip()) == 0:
                    num = input("Id не может быть пустой строкой!")
                if self.check_sql_injections(num):
                    print("Недопустимый ввод!")
                    return '1'
                if num == "0":
                    return "1"
                person = PeopleTable().find_by_position(int(num))
                if not person:
                    print("Нет такого человека!")
                else:
                    self.person_id = int(person[0])
                    self.person_obj = person
                    break
        print("Выбран человек: " +
                self.person_obj[1] + " " + self.person_obj[2] + " " +
self.person_obj[3])
        print("Тип\tСерия\tНомер\tКем выдан\tДата выдачи")
        lst = DocumentsTable().all_by_person_id(self.person_id)
        for i in lst:
```

```
            print(i[1] + "\t" + i[2] + "\t" + i[3] + "\t" + i[4] +  "\t\t" +
i[5])
        print(self.documents_menu)
        return self.read_next_step()


    def show_add_document(self):
        tipe = self.input_check('тип документа', length=32)
        if tipe is None:
            return '1'

        while True:
            series = self.input_check("серия документа", 15)
            if series is None:
                return '1'
            if not series.isnumeric():
                print("Некорректный ввод!")
            break

        while True:
            num = self.input_check('номер документа', 15)
            if num is None:
                return '1'
            if not num.isnumeric():
                print("Некорректный ввод!")
            break

        organ = self.input_check('кем выдан', 64)
        if organ is None:
            return '1'

        while True:
            date = self.input_check('дата выдачи', 10)
            if date is None:
                return '1'
            try:
                dt = datetime.strptime(date, "%d.%m.%Y")
            except:
                print('Некорректный ввод. Дата должна быть в формате
ДД.ММ.ГГГГ!')
            else:
                break

        exist = DocumentsTable().check_field_exist(
            ['person_id', 'series', "number"], [self.person_id, series, num])
        print(self.documents_menu)
        if exist:
            print("Нарушен Primary Key, запись не вставлена")
            return self.read_next_step()
        DocumentsTable().insert_one([date,  num, self.person_id,  organ, series,
tipe])
```

```
        return self.read_next_step()


    def delete_document(self):
        return self.delete_by_field(DocumentsTable, self.documents_menu,
'person_id')


m = Main()
m.main_cycle()
```

Файл Dbtable

```
from dbconnection import *
from psycopg2 import sql

class DbTable:
    dbconn = None

    def __init__(self):
        return

    def table_name(self):
        return self.dbconn.prefix + "table"

    def columns(self):
        return {"test": ["integer", "PRIMARY KEY"]}

    def column_names(self):
        #return sorted(self.columns().keys(), key = lambda x: x)
        return self.columns().keys()

    def primary_key(self):
        return ['id']

    def column_names_without_id(self):
        res = sorted(self.columns().keys(), key = lambda x: x)
        if 'id' in res:
            res.remove('id')
        return res

    def table_constraints(self):
        return []

    def create(self):
        arr = [k + " " + " ".join(v) for k, v in self.columns().items()]
        sql = f"CREATE TABLE {self.table_name()} ({', '.join(arr +
self.table_constraints())})"
        cur = self.dbconn.conn.cursor()
```

```
        cur.execute(sql)
        self.dbconn.conn.commit()

    def drop(self):
        sql = "DROP TABLE IF EXISTS " + self.table_name()
        cur = self.dbconn.conn.cursor()
        cur.execute(sql)
        self.dbconn.conn.commit()

    def insert_one(self, vals):
        sql = f"""INSERT INTO {self.table_name()} ({",
".join(self.column_names_without_id())}) VALUES({", ".join(["%s" for _ in
range(len(vals))])}) """
        cur = self.dbconn.conn.cursor()
        cur.execute(sql, tuple(vals))
        self.dbconn.conn.commit()

    def first(self):
        sql = f"SELECT * FROM {self.table_name()} ORDER BY {',
'.join(self.primary_key())}"
        cur = self.dbconn.conn.cursor()
        cur.execute(sql)
        return cur.fetchone()

    def last(self):
        sql = f"SELECT * FROM {self.table_name()} ORDER BY {', '.join([x + '
DESC' for x in self.primary_key()])}"
        cur = self.dbconn.conn.cursor()
        cur.execute(sql)
        return cur.fetchone()

    def all(self):
        sql = f"SELECT * FROM  {self.table_name()} ORDER BY {',
'.join(self.primary_key())}"
        cur = self.dbconn.conn.cursor()
        cur.execute(sql)
        return cur.fetchall()

    def check_field_exist(self, field, field_val):
        cur = self.dbconn.conn.cursor()
        isList = isinstance(field, list) and isinstance(field_val, list)
        fields = field if isList else [field]
        vals = field_val if isList else [field_val]
        cur.execute(
            sql.SQL(
                f"""SELECT * FROM {self.table_name()}
                    WHERE {" AND ".join(['{} = %s' for _ in
range(len(fields))])}"""
            ).format(*[sql.Identifier(x) for x in fields]), (*vals, ))
        rows = cur.fetchall()
        return bool(rows and len(rows) > 0)
```

```
    def delete_one_by_field(self, field, field_val):
        is_exist = self.check_field_exist(field, str(field_val))
        if (is_exist):
            self.dbconn.conn.cursor().execute(
                sql.SQL(
                    f"""DELETE FROM {self.table_name()}
                        WHERE {{}} = %s"""
                ).format(sql.Identifier(field)), (field_val,))
            self.dbconn.conn.commit()
        return is_exist
```

Файл Dbconnection

```
import psycopg2

class DbConnection:

    def __init__(self, config):
        self.path = config.dbfilepath
        self.prefix = config.dbtableprefix
        self.conn = psycopg2.connect(dbname=config.dbname,
                                     user=config.user, password=config.password,
host=config.host)

    def __del__(self):
        if self.conn:
            self.conn.close()

    def test(self):
        cur = self.conn.cursor()
        cur.execute("CREATE TABLE test(test integer)")
        cur.execute("INSERT INTO test(test) VALUES(1)")
        self.conn.commit()
        cur.execute("SELECT * FROM test")
        result = cur.fetchall()
        cur.execute("DROP TABLE test")
        self.conn.commit()
        return (result[0][0] == 1)
```

Файл project_config

```
import yaml

class ProjectConfig:
    """Класс считывает базовые настройки из файла config.yaml"""

    def __init__(self):
        with open('config.yaml') as f:
```

```
            config = yaml.safe_load(f)
            self.dbfilepath = config['dbfilepath']
            self.dbtableprefix = config['dbtableprefix']
            self.dbname = config['dbname']
            self.user = config['user']
            self.host = config['host']
            self.password = config['password']


if __name__ == "__main__":
    x = ProjectConfig()
    print(x.dbfilepath)
```

Файл documents_table

```
from dbtable import *


class DocumentsTable(DbTable):
    def table_name(self):
        return self.dbconn.prefix + "documents"

    def columns(self):
        return {"person_id": ["integer", f"REFERENCES
{self.dbconn.prefix}people(id) ON DELETE CASCADE"], #3
                "type": ["varchar(32)", "NOT NULL"], #6
                "series": ["varchar(15)"], #5
                "number": ["varchar(15)", "NOT NULL"], #2
                "regulator": ["varchar(64)", "NOT NULL"], #4
                "date": ["varchar(10)", "NOT NULL"] #1
                }

    def primary_key(self):
        return ['person_id', "series", "number"]

    def table_constraints(self):
        return ["PRIMARY KEY(person_id, series, number)"]

    def all_by_person_id(self, pid):
        sql = f"SELECT * FROM {self.table_name()} WHERE person_id = %s ORDER BY
{', '.join(self.primary_key())}"
        cur = self.dbconn.conn.cursor()
        cur.execute(sql, str(pid))
        return cur.fetchall()
```

Файл people_table

```
from dbtable import *
```

```
class PeopleTable(DbTable):
    def table_name(self):
        return self.dbconn.prefix + "people"

    def columns(self):
        return {"id": ["SERIAL", "PRIMARY KEY"], #2
                "last_name": ["varchar(32)", "NOT NULL"], #3
                "first_name": ["varchar(32)", "NOT NULL"], #1
                "second_name": ["varchar(32)"]} #4

    def find_by_position(self, num):
        sql = f"SELECT * FROM {self.table_name() } WHERE id = %s"
        cur = self.dbconn.conn.cursor()
        cur.execute(sql, [num])
        return cur.fetchone()

    def pst_read(self):
        elems = 2
        cur_page = 0
        print(f'{cur_page + 1} страница таблицы {self.table_name()}')
        [print(i) for i in self.all()[:elems]]
        k = input('Введите:\n1 - для перехода на введенную страницу;\n2 - для
переход на следующую страницу;'
                          '\n3 - для возврата на предыдущую страницу;\nиное -
выход')
        k = int(k)
        while True:
            if k == 1:
                while True:
                    try:
                        t = input('Введите номер страницы:\n ')
                        t = int(t)
                        cur_page = t - 1
                        print(f'{cur_page + 1} страница таблицы
{self.table_name()}')
                        [print(i) for i in self.all()[cur_page * elems :(cur_page
+ 1) * elems]]
                    except:
                        print("Некорректный ввод!")
                    break
            elif k == 2:
                cur_page = cur_page + 1
                print(f'{cur_page + 1} страница таблицы {self.table_name()}')
                [print(i) for i in self.all()[cur_page * elems : (cur_page + 1) *
elems]]
            elif k == 3:
                print(f'{cur_page + 1} страница таблицы {self.table_name()}')
                cur_page = cur_page - 1
                [print(i) for i in self.all()[cur_page * elems : (cur_page + 1) *
elems]]
            else: return
```

```
            k = input('Введите:\n1 - для перехода на введенную страницу;\n2 - для
переход на следующую страницу;'
                      '\n3 - для возврата на предыдущую страницу;\nиное -
выход')
            k = int(k)
```

Файл phones_table

```
from dbtable import *

class PhonesTable(DbTable):
    def table_name(self):
        return self.dbconn.prefix + "phones"

    def columns(self):
        return {"person_id": ["integer", f"REFERENCES
{self.dbconn.prefix}people(id)"],
                "phone": ["varchar(12)", "NOT NULL"]}

    def primary_key(self):
        return ['person_id', 'phone']

    def table_constraints(self):
        return ["PRIMARY KEY(person_id, phone)"]

    def all_by_person_id(self, pid):
        sql = f"SELECT * FROM {self.table_name()} WHERE person_id = %s ORDER BY
{', '.join(self.primary_key())}"
        cur = self.dbconn.conn.cursor()
        cur.execute(sql, str(pid))
        return cur.fetchall()
```