Экзамен Базы данных и экспертные системы Монастырский С21-703

Лабораторная работа 5

```
--0
DROP TRIGGER IF EXISTS place_restrictions_trigger ON "C21-703-7".product CASCADE;

CREATE OR REPLACE FUNCTION place_restrictions_trigger_func() RETURNS trigger as $$
BEGIN
if(TG_OP = 'DELETE') then
if(((SELECT spaces_left+1 from "C21-703-7"."Shelf" where shelf_id = old.shelf_id)
<=
    (SELECT max_spaces from "C21-703-7"."Shelf" where shelf_id = old.shelf_id)))
  then
UPDATE "C21-703-7"."Shelf" SET spaces_left = spaces_left+1 where shelf_id =
old.shelf_id;
UPDATE "C21-703-7"."Shelf" SET weight_left = weight_left+old.weight where shelf_id
= old.shelf_id;
else
RAISE EXCEPTION 'Нарушены правила использования мест на полке: %', old.shelf_id;
END IF;
elsif (TG_OP = 'INSERT') then
if(((SELECT spaces_left-1 from "C21-703-7"."Shelf" where shelf_id = new.shelf_id)
>= 0))
 then

UPDATE "C21-703-7"."Shelf" SET spaces_left = spaces_left-1 where shelf_id =
new.shelf_id;
UPDATE "C21-703-7"."Shelf" SET weight_left = weight_left-new.weight where shelf_id
= new.shelf_id;
else
RAISE EXCEPTION 'Нарушены правила использования мест на полке: %', new.shelf_id;
END IF;

elsif (TG_OP = 'UPDATE') then
UPDATE "C21-703-7"."Shelf" SET spaces_left = spaces_left +1 WHERE shelf_id =
old.shelf_id;
UPDATE "C21-703-7"."Shelf" SET spaces_left = spaces_left -1 WHERE shelf_id =
new.shelf_id;
UPDATE "C21-703-7"."Shelf" SET weight_left = weight_left - new.weight WHERE
shelf_id = new.shelf_id;
```

```
UPDATE "C21-703-7"."Shelf" SET weight_left = weight_left - old.weight WHERE
shelf_id = old.shelf_id;
END IF;
RETURN new;
END;
$$ language plpgsql;

CREATE OR REPLACE TRIGGER place_restrictions_trigger BEFORE INSERT OR DELETE OR
UPDATE ON "C21-703-7"."product"
FOR EACH ROW
EXECUTE PROCEDURE place_restrictions_trigger_func()


UPDATE "C21-703-7"."product" SET weight = 15;

SELECT * FROM "C21-703-7"."Shelf";
INSERT INTO "C21-703-7"."product"
Values(nextval('pid_generator'),110,120,130,now(),1,60,30,20,50,5,slot_finder(5),10
);
INSERT INTO "C21-703-7"."product"
Values(nextval('pid_generator'),110,120,130,now(),1,60,30,20,50,5,slot_finder(5),1)
;
INSERT INTO "C21-703-7"."product"
Values(nextval('pid_generator'),110,120,130,now(),1,60,30,20,50,5,slot_finder(5),1)
;
INSERT INTO "C21-703-7"."product"
Values(nextval('pid_generator'),110,120,130,now(),1,60,30,20,50,5,slot_finder(5),1)
;
INSERT INTO "C21-703-7"."product"
Values(nextval('pid_generator'),110,120,130,now(),1,60,30,20,50,5,slot_finder(5),1)
;
SELECT * FROM "C21-703-7"."Shelf";

INSERT INTO "C21-703-7"."product"
Values(nextval('pid_generator'),300,300,300,now(),2,60,30,20,50,2,slot_finder(2),59
9.99);
SELECT * FROM "C21-703-7"."Shelf";
UPDATE "C21-703-7".product SET shelf_id = 3 where shelf_id = 2;
DELETE FROM "C21-703-7".product WHERE shelf_id = 5;
--1
CREATE OR REPLACE FUNCTION product_finder(client_n varchar(255),cdate date) RETURNS
integer as $$
BEGIN
return (SELECT count(product_id) from "C21-703-7"."product" p JOIN "C21-703-
7"."Contract" c on(p.contract_id = c.contract_id)
 LEFT JOIN "C21-703-7"."Client" cl on (cl.client_id = c.client_id)
```

```
 where (name = client_n and expiration_date < cdate));
 END;
$$ language plpgsql;

SELECT product_finder('PAO SBERBANK','29.01.2003');


--2
CREATE OR REPLACE FUNCTION max_parameters_step(numeric[], numeric[]) RETURNS
numeric[] AS $$
DECLARE
res numeric[];
BEGIN
IF $1[1] > $2[1] THEN
res[1] := $1[1];
ELSE
res[1] := $2[1];
END IF;
IF $1[2] > $2[2] THEN
res[2] := $1[2];
ELSE
res[2] := $2[2];
END IF;
IF $1[3] > $2[3] THEN
res[3] := $1[3];
ELSE
res[3] := $2[3];
END IF;
RETURN res;
END
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION max_parameters_final(numeric[]) RETURNS text AS $$
SELECT (to_char($1[1],'99999999D999') || 'x' || to_char($1[2],'99999999D999') ||
'x' ||to_char($1[3],'9999999999D999'));
$$ LANGUAGE sql;

CREATE OR REPLACE AGGREGATE maxpam(numeric[]) (
sfunc = max_parameters_step,
stype = numeric[],
initcond = '{0, 0, 0}',
finalfunc = max_parameters_final
);

select maxpam(ARRAY[p.height,p.width,p.length]) FROM "C21-703-7"."product" p
```

```sql
--3
drop view client_product_view;
CREATE VIEW client_product_view AS
SELECT c.name AS client_name, c.client_id as client_id, c.requisites, p.product_id,
p.width, p.height, p.length, p.unpacking_date, p.shelf_id, p.slot_id, p.weight
FROM "C21-703-7"."Client" c
INNER JOIN "C21-703-7"."Contract" ct ON c.client_id = ct.client_id
INNER JOIN "C21-703-7"."product" p ON ct.contract_id = p.contract_id;



CREATE OR REPLACE FUNCTION update_view() RETURNS TRIGGER AS $$
BEGIN
UPDATE "C21-703-7"."Client" SET requisites = NEW.requisites WHERE id= OLD.id;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER clinet_product_view
INSTEAD OF UPDATE ON client_product_view FOR EACH ROW EXECUTE PROCEDURE
update_view();

UPDATE client_product_view SET requisites = 'OOO CHTOTO' WHERE client_id = 1;



--4
CREATE FUNCTION init()
RETURNS VOID
AS $$
BEGIN
DROP TABLE IF EXISTS queue;
    CREATE TABLE queue (
                id SERIAL PRIMARY KEY,
                data VARCHAR(64) NOT NULL,
                inserted_at TIMESTAMP NOT NULL DEFAULT NOW()
            );
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION enqueue(new_data VARCHAR(64))
RETURNS VOID
AS
$$
BEGIN
```

```
    INSERT INTO queue (data) VALUES (new_data);
END;
$$
LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION dequeue()
RETURNS VOID
AS
$$
BEGIN
    DELETE FROM queue WHERE id = (SELECT min(id) FROM queue);
END;
$$
LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION empty()
RETURNS VOID
AS
$$
BEGIN
    DELETE FROM queue;
END;
$$
LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION top()
RETURNS VARCHAR(64)
AS
$$
BEGIN
    RETURN (SELECT data FROM queue ORDER BY id LIMIT 1);
END;
$$
LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION tail()
RETURNS VARCHAR(64)
AS
$$
BEGIN
    RETURN (SELECT data FROM queue ORDER BY id ASC LIMIT 1);
END;
$$
LANGUAGE plpgsql;
```

```
--test


select init();
select init();
select enqueue('Mathematics');
select enqueue('Physics');
select enqueue('English');
select enqueue('Biology');
select enqueue('Social studies');
select * from queue;
select dequeue();
select top();
select tail();
select dequeue();
select dequeue();
select dequeue();
select dequeue();
select dequeue();
select dequeue();
```

Лабораторная работа 6

```
# Установка соединения с базой данных
# (параметры передаются через класс конфиг).
import logging
from datetime import date
from os import sep
import psycopg2
from psycopg2.extras import LoggingConnection
LOG_FILE ="log" + sep + str(date.today()) + ".log"

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)
FH = logging.FileHandler(LOG_FILE)
basic_formater = logging.Formatter('%(asctime)s : [%(levelname)s] : %(message)s')
FH.setFormatter(basic_formater)
logger.addHandler(FH)
logger.propagate = False
```

```
class DbConnection:

    def __init__(self, config):
        self.dbname = config.dbname
        self.user = config.user
        self.password = config.password
        self.host = config.host
        self.prefix = config.dbtableprefix
        self.port = config.port
        self.conn = psycopg2.connect(dbname = self.dbname,
                                     user = self.user,
                                     password = self.password,
                                     host = self.host,
                                     port=self.port,
                                     connection_factory=LoggingConnection)
        self.conn.initialize(logger)
        self.logger = logger

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

    def test(self):
        cur = self.conn.cursor()
        cur.execute("DROP TABLE IF EXISTS test CASCADE")
        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)

# Базовые действия с таблицами

from dbconnection import *

class DbTable:
    dbconn = None

    def __init__(self)->None:
        return

    def table_name(self)->str:
```

```
        """
        префикс + таблица
        """
        return self.dbconn.prefix + "table"

    def columns(self)->dict:
        """
        колонки для инициализатора + констреинты к ней
        """
        return {"test": ["integer", "PRIMARY KEY"]}

    def column_names(self)->list:
        """
        все колонки
        """
        return self.columns().keys()

    def primary_key(self)->list:
        """
        ключевое поле
        """
        return ['id']

    def column_names_without_id(self)->list:
        """
        возвращает список колонок не включая ключ
        """
        res = list(self.columns().keys())
        for col in self.primary_key():
            res.remove(col)
        return res

    def table_constraints(self)->list:
        """
        Список общих констреинтов таблицы
        """
        return []

    def create(self)->None:
        """
        Создать таблицу в БД
        """
        sql = "CREATE TABLE " + self.table_name() + "("
        arr = [k + " " + " ".join(v) for k, v in self.columns().items()]
        #print(arr)
        sql += ", ".join(arr + self.table_constraints())
```

```
        sql += ")"
        #print(sql)
        cur = self.dbconn.conn.cursor()
        cur.execute(sql)
        self.dbconn.conn.commit()
        return

    def drop(self)->None:
        """
        удалить таблицу
        """
        sql = "DROP TABLE IF EXISTS " + self.table_name()
        cur = self.dbconn.conn.cursor()
        cur.execute(sql)
        self.dbconn.conn.commit()
        return

    def insert_one(self, vals:list)->None:
        """
        Добавить запись в таблицу
        """
        for i in range(0, len(vals)):
            if type(vals[i]) == str:
                vals[i] = "'" + vals[i] + "'"
            else:
                vals[i] = str(vals[i])
        sql = "INSERT INTO " + self.table_name() + "("
        sql += ",".join(self.column_names_without_id()) + ") VALUES("
        sql +=  "(%s)," * (len(vals)-1) + "(%s)" + ")"
        #print(sql)
        cur = self.dbconn.conn.cursor()
        #print(sql)
        cur.execute(sql, vals)
        self.dbconn.conn.commit()
        return

    def first(self)->list:
        """
        Выбрать первую запись из таблицы
        """
        sql = "SELECT * FROM " + self.table_name()
        sql += " ORDER BY "
        sql += ", ".join(self.primary_key())
        cur = self.dbconn.conn.cursor()
        cur.execute(sql)
        return cur.fetchone()
```

```
    def last(self)->list:
        """
        Выбрать последнюю запись из таблицы
        """
        sql = "SELECT * FROM " + self.table_name()
        sql += " ORDER BY "
        sql += ", ".join([x + " DESC" for x in self.primary_key()])
        cur = self.dbconn.conn.cursor()
        cur.execute(sql)
        return cur.fetchone()

    def all(self)->list:
        """
        выбрать все записи из таблицы
        """
        sql = "SELECT * FROM " + self.table_name()
        sql += " ORDER BY "
        sql += ", ".join(self.primary_key())
        cur = self.dbconn.conn.cursor()
        cur.execute(sql)
        return cur.fetchall()


    def create_list_of_ids(self)-> list:
        """
        список айдишников в таблице
        """
        sql = f"SELECT {self.primary_key()[0]} FROM " + self.table_name()
        cur = self.dbconn.conn.cursor()
        cur.execute(sql)
        ids = [x[0] for x in cur.fetchall()]
        #print(ids)
        return ids



PG_INT_MAX = 2147483647
PG_INT_MIN = -2147483648

NUMERIC7_0_MAX = 1e7-1
NUMERIC7_0_MIN = -1e6-1

NUMERIC7_2_MAX = 1e4-0.001
NUMERIC7_2_MIN = -1e3-0.001
```

```
NUMERIC7_2_MAX = 1e5-0.001
NUMERIC7_2_MIN = -1e4-0.001


NUMERIC7_2_MAX = 1e2-0.001
NUMERIC7_2_MIN = -1e1-0.001

VARCHAR_MAX = 50

# Загрузка настроек проекта (в данном случае только настроек соединения с БД)
# из файла config.yaml.
import yaml
import os

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

    def __init__(self):
        with open("config"+os.sep +'config_notebook.yaml') as f:
            config = yaml.safe_load(f)
            self.dbname = config['dbname']
            self.user = config['user']
            self.password = config['password']
            self.host = config['host']
            self.port=config['port']
            self.dbtableprefix = config['dbtableprefix']

#print(self.dbname,self.user,self.password,self.host,self.dbtableprefix,sep = '\n')

# Этот метод запускается только, если запускать
# данный файл, а не подключать его.
if __name__ == "__main__":
    x = ProjectConfig()
    print(x)
```

```
from dbtable import *
import colorama
from colorama import Fore, Back, Style
import pglimits
colorama.init()
class RoomTable(DbTable):
    def table_name(self) -> str:
        """
    Возвращает строку схема + название таблицы
```

```
    """
        return self.dbconn.prefix + "Room"


    def columns(self)->dict:
        """
        Возвращает список полей + ограничения целостности на конкретные поля
        """
        return {
        "id":["serial","PRIMARY KEY"],
        "name": ["character varying(50)", "NOT NULL"],
        "space": ["numeric(7,2)", "NOT NULL"],
        "space_left":["numeric(7,2)", "NOT NULL"],
        "min_humidity":["numeric(8,2)","NOT NULL"],
        "max_humidity":["numeric(8,2)" ,"NOT NULL"],
        "min_temp": ["numeric(5,2)", "NOT NULL"],
        "max_temp":["numeric(5,2)","NOT NULL"]}


    def table_constraints (self)->list:
        """
        Возвращает список общих ограничений целостности
        """
        return ([
        "CONSTRAINT uni_room_name UNIQUE (name)",
           "CONSTRAINT positive_volume_left_room CHECK(space_left >0)",
           "CONSTRAINT positive_volume_room CHECK(space >0)",
           "CONSTRAINT volume_left_le_volume CHECK(space_left <= space)",
           "CONSTRAINT hu_max_in_interval CHECK (max_humidity <= 100 and
max_humidity >= 0)",
           "CONSTRAINT hu_min_in_interval CHECK (min_humidity <= 100 and
min_humidity >= 0)",
       "CONSTRAINT temp_check CHECK (min_temp <= max_temp)",
       "CONSTRAINT hu_check CHECK (min_humidity <= max_humidity)"

       ])



    def delete_room(self)->None:
        id_ = None
        """
        Отрабатывает удаление комнаты
        """
        while not ((type(id_) == int) and (id_ in self.create_list_of_ids())):
            try:
```

```
                id_ = int(input(Fore.YELLOW +"Введите номер комнаты, которую хотите
удалить (введите -1 для отмены): " + Style.RESET_ALL).strip())
                if id_ == -1:
                    return
                if id_ not in self.create_list_of_ids():
                    raise ValueError
            except ValueError as e:
                print(Fore.RED +"Введите номер по списку, введеное значение - не
число" + Style.RESET_ALL)
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
        sql = "DELETE FROM " + self.table_name()
        sql += f" WHERE id = (%s)"
        cur = self.dbconn.conn.cursor()
        id_ = str(id_)
        cur.execute(sql, (id_,))
        self.dbconn.conn.commit()
        self.show_rooms()


    def show_rooms(self)->None:
        """
        отобразить список комнат в терминал
        """
        menu = """Просмотр списка комнат!
№\tИмя\tОбъем\t"""
        print(Fore.YELLOW + menu + Style.RESET_ALL)
        lst = self.all()
        for i in lst:
            print(Fore.GREEN+str(i[0])+Style.RESET_ALL + "\t" + str(i[1]) + "\t" +
str(i[2]))
        menu = Fore.YELLOW +"""Дальнейшие операции:
    """+Style.RESET_ALL +Fore.GREEN + str(0) + Style.RESET_ALL+"""  - возврат в
главное меню;
    """+Fore.GREEN + str(3) + Style.RESET_ALL+"""  - добавление новой комнаты;
    """+Fore.GREEN + str(4) + Style.RESET_ALL+"""  - удаление комнаты;
    """+Fore.GREEN + str(5) + Style.RESET_ALL+"""  - просмотр стеллажей комнаты;
    """+Fore.GREEN + str(6) + Style.RESET_ALL+"""  - редактировать комнату
    """+Fore.GREEN + str(9) + Style.RESET_ALL+"""  - выход."""
        print(menu)
        return


    def __call_creation_wizard(self)->list:
        name = ""
        space = None
        minh=maxh = None
```

```
        mint=maxt = None
        """
        Запуск мастера создания комнаты
        """
        while not (0< len(name) <= pglimits.VARCHAR_MAX):
            try:
                name = input(Fore.YELLOW+ "Введите название комнаты или пустую
строку для отмены: " + Style.RESET_ALL)
                if not(len(name)):
                    return
                if (len(name) > pglimits.VARCHAR_MAX):
                    raise ValueError
            except ValueError:
                print(Fore.RED + "Строка слишком большая" + Style.RESET_ALL)

        while not ((type(space) == float) and (pglimits.NUMERIC7_2_MIN <= space <=
pglimits.NUMERIC7_2_MAX)):
            try:
                space = float(input(Fore.YELLOW+ "Введите объем комнаты для отмены
введите 0: " + Style.RESET_ALL))
                if not space:
                    return
                if (space < 0) or not (pglimits.NUMERIC7_2_MIN <= space <=
pglimits.NUMERIC7_2_MAX):
                    raise ValueError
            except ValueError as e:
                print(Fore.RED+"Введено неверное число" + Style.RESET_ALL)
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)


        while not ((type(minh)==float) and (type(maxh)==float) and (0< minh <= 100)
and (0 < maxh <= 100) and minh <= maxh):
            try:
                minh = float(input(Fore.YELLOW+ "Введите минимальную влажность для
отмены введите 0: " + Style.RESET_ALL))
                if not(minh):
                    return
                maxh = float(input(Fore.YELLOW+ "Введите максимальную влажность для
отмены введите 0: " + Style.RESET_ALL))
                if not (maxh):
                    return
                if not((0< minh <= 100) and (0 < maxh <= 100) and minh <= maxh):
                    raise ValueError
            except ValueError as e:
                print(Fore.RED+ "Одна или обе из влажностей введена(ы)
неверно"+Style.RESET_ALL)
```

```
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)

        while not ((type(mint)==float) and (type(maxt)==float) and (0< mint <= 100)
and (0 < maxt <= 100) and mint <= maxt):
            try:
                mint = float(input(Fore.YELLOW+ "Введите минимальную температуру
для отмены введите 0: " + Style.RESET_ALL))
                if not mint:
                    return
                maxt = float(input(Fore.YELLOW+ "Введите максимальную температуру
для отмены введите 0: " + Style.RESET_ALL))
                if not maxt:
                    return
                if not((0< mint <= 100) and (0 < maxt <= 100) and mint <= maxt):
                    raise ValueError
            except ValueError as e:
                print(Fore.RED + "Одна или обе из температур введена(ы)
неверно"+Style.RESET_ALL)
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
        return [name,space,space,minh,maxh,mint,maxt]




    def add_rooms(self)-> None:
        """
        обработчик создания комнаты
        """
        data = self.__call_creation_wizard()
        if type(data) == list:
            self.insert_one(data)
            print(Fore.GREEN + "Комната создана"+ Style.RESET_ALL)


    def edit_check(self,id_:int, col_2edit: int, new_data:str)-> bool:
        """
        Контроль за соблюдением ограничений целостности при изменении полей в
методе edit_room
        """
        if(col_2edit == 0):
            try:
                if len(new_data) > pglimits.VARCHAR_MAX:
                    raise ValueError
            except ValueError:
                    print(Fore.RED + "Строка слишком большая" + Style.RESET_ALL)
```

```
                    new_data = input(Fore.YELLOW + "Введите заново: "+
Style.RESET_ALL)
                    return self.edit_check(id_,col_2edit,new_data)
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            cur.execute(sql, [new_data, str(id_,)])
            self.dbconn.conn.commit()
            return True
        elif (col_2edit == 1):
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            try:
                sql_ = f"SELECT {self.column_names_without_id()[col_2edit+1]} FROM
{self.table_name()} WHERE {self.primary_key()[0]} = (%s)"
                cur = self.dbconn.conn.cursor()
                cur.execute(sql_, (str(id_),))
                recived = list(cur.fetchone())[0]

                if (not(float(new_data) >= recived) or (float(new_data) <= 0) or
                    not(pglimits.NUMERIC7_2_MIN<= float(new_data) <=
pglimits.NUMERIC7_2_MAX)):
                    raise ValueError
            except ValueError as e:
                print(Fore.RED +"Введено неверное значение" + Style.RESET_ALL)
                new_data = input(Fore.YELLOW + "Введите заново: "+ Style.RESET_ALL)
                return self.edit_check(id_,col_2edit,new_data)

                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)

            cur.execute(sql, [new_data, str(id_)])
            self.dbconn.conn.commit()
            return True
        elif (col_2edit == 3):
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            try:
                d = float(new_data)
                min_ = f"SELECT {self.column_names_without_id()[-3]} FROM " +
self.table_name() + f" WHERE {self.primary_key()[0]} = {str(id_)}"
                cur.execute(min_)
```

```
                r = float(cur.fetchone()[0])

                if not ((0 <= d <= 100)) or (d > r):
                    raise ValueError
            except ValueError as e:
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
                print(Fore.RED +"Введено неверное значение" + Style.RESET_ALL)
                new_data = input(Fore.YELLOW + "Введите заново: "+ Style.RESET_ALL)
                return self.edit_check(id_,col_2edit,new_data)
            cur.execute(sql, [new_data, str(id_)])
            self.dbconn.conn.commit()
            return True
        elif (col_2edit == 4):
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            try:
                d = float(new_data)
                min_ = f"SELECT {self.column_names_without_id()[-4]} FROM " +
self.table_name() + f" WHERE {self.primary_key()[0]} = {str(id_)}"
                cur.execute(min_)
                r = float(cur.fetchone()[0])

                if not ((0 <= d <= 100)) or (d < r):
                    raise ValueError
            except ValueError as e:
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
                print(Fore.RED +"Введено неверное значение" + Style.RESET_ALL)
                new_data = input(Fore.YELLOW + "Введите заново: "+ Style.RESET_ALL)
                return self.edit_check(id_,col_2edit,new_data)
            cur.execute(sql, [new_data, str(id_)])
            self.dbconn.conn.commit()
            return True
        elif (col_2edit == 5):
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            try:
                d = float(new_data)
                min_ = f"SELECT {self.column_names_without_id()[-1]} FROM " +
self.table_name() + f" WHERE {self.primary_key()[0]} = {str(id_)}"
                cur.execute(min_)
                r = float(cur.fetchone()[0])
                if not ((0 <= d <= 100)) or (d > r):
```

```
                    raise ValueError
            except ValueError as e:
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
                print(Fore.RED +"Введено неверное значение" + Style.RESET_ALL)
                new_data = input(Fore.YELLOW + "Введите заново: "+ Style.RESET_ALL)
                return self.edit_check(id_,col_2edit,new_data)
            cur.execute(sql, [new_data, str(id_)])
            self.dbconn.conn.commit()
            return True
        elif (col_2edit == 6):
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            try:
                d = float(new_data)
                min_ = f"SELECT {self.column_names_without_id()[-2]} FROM " +
self.table_name() + f" WHERE {self.primary_key()[0]} = {str(id_)}"
                cur.execute(min_)
                r = float(cur.fetchone()[0])
                if not ((0 <= d <= 100)) or (d < r):
                    raise ValueError
            except ValueError as e:
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
                print(Fore.RED +"Введено неверное значение" + Style.RESET_ALL)
                new_data = input(Fore.YELLOW + "Введите заново: "+ Style.RESET_ALL)
                return self.edit_check(id_,col_2edit,new_data)
            cur.execute(sql, [new_data, str(id_)])
            self.dbconn.conn.commit()
            return True
        else:
            print(Fore.RED+"Ошибка, затронут неверный столбец, возможно
неизменяемый"+Style.RESET_ALL)
            return False



    def edit_room(self):
        inp = None
        col_2edit = None
        """
        Изменение параметров существующей сущности комнаты
        """
        self.show_rooms()
        while not (inp in self.create_list_of_ids()):
            try:
```

```
                inp = int(input(Fore.YELLOW+"Выберите комнату которую хотите
изменить: "+Style.RESET_ALL))
                if inp not in self.create_list_of_ids():
                    raise ValueError
            except ValueError as e:
                print(Fore.RED+"Введите правильное число"+Style.RESET_ALL)
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
        sql = f"SELECT * FROM {self.table_name()} WHERE {self.primary_key()[0]} =
(%s)"
        cur = self.dbconn.conn.cursor()
        cur.execute(sql, (str(inp),))
        recived = list(cur.fetchone())[1:]
        cols = self.column_names_without_id()
        data = list(zip(cols,recived))
        for col in enumerate(data):
            print(col[0],col[1], sep = "\t")
        while not(type(col_2edit) == int and (0 <= col_2edit < len(data))):
            try:
                col_2edit = int(input(Fore.YELLOW+"Введите номер поля, который
хотите изменить: " + Style.RESET_ALL))
                if not(0 <= col_2edit < len(data)):
                    raise ValueError
            except ValueError as e:
                print(Fore.RED+"Введено неверное число"+Style.RESET_ALL)
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
        new_data = input(Fore.YELLOW+"Введите новое значение поля:
"+Style.RESET_ALL)
        if self.edit_check(inp,col_2edit,new_data):
            print(Fore.GREEN+"Изменения применены"+Style.RESET_ALL)
            return
        print(Fore.RED+"Во внесении изменений отказно, неверное новое
значение"+Style.RESET_ALL)
        return


from dbtable import *
from room_table import RoomTable
import colorama
import pglimits
from colorama import Fore, Back, Style
colorama.init()


class ShelfTable(DbTable):
    def table_name(self)->str:
        """
```

```
        Возвращает строку префикс + полка
        """
        return self.dbconn.prefix + "Shelf"
    def columns(self)->dict:
        """
        возвращает колонки + локальные ограничения целостности
        """
        return {
        "shelf_id":["serial","PRIMARY KEY"],
        "room_id":["integer",f'REFERENCES {self.dbconn.prefix}Room ON DELETE
CASCADE'],
        "max_spaces":["integer","NOT NULL"],
        "spaces_left":["integer", "NOT NULL"],
        "slot_w":["numeric(7,0)", "NOT NULL"],
        "slot_h":["numeric(7,0)","NOT NULL"],
        "slot_l":["numeric(7,0)", "NOT NULL"],
        "max_weight":["numeric(7,2)", "NOT NULL"],
        "weight_left":["numeric(7,2)", "NOT NULL"]
        }

    def table_constraints(self)->list:
        """
        Возвращает общие ограничения целостности
        """
        return[
        "CONSTRAINT positive_size_slot_w_shelf CHECK(slot_w >0)",
        "CONSTRAINT positive_size_slot_h_shelf CHECK(slot_h >0)",
        "CONSTRAINT positive_size_slot_l_shelf CHECK(slot_l >0)",
        "CONSTRAINT positive_weight_shelf CHECK(max_weight >0)",
        "CONSTRAINT positive_weight_left_shelf CHECK(weight_left >0)",
        "CONSTRAINT wight_left_le_weight CHECK(weight_left <= max_weight)"
        ]
    def primary_key(self)->list:
        """
        Возвращает список ключевых полей
        """
        return ['shelf_id']

    def all_by_room_id(self, room_id:int):
        rt = RoomTable()
        """
        Возвращает список полок по айди комнаты
        """
        if room_id not in rt.create_list_of_ids():
            print(Fore.RED +"неверное значение, Выберите существующее" +
Style.RESET_ALL)
```

```
        sql = "SELECT * FROM " + self.table_name()
        sql += " WHERE room_id = (%s)"
        sql += " ORDER BY "
        sql += ", ".join(self.primary_key())
        cur = self.dbconn.conn.cursor()
        cur.execute(sql, (str(room_id),))
        return cur.fetchall()

    def delete_shelf(self):
        """
        Удаление полки
        """
        try:
            id_ = int(input(Fore.YELLOW+"Введите номер полки, которую хотите
удалить: "+Style.RESET_ALL).strip())
            if not id_ in self.create_list_of_ids():
                raise ValueError
        except ValueError as e:
            print(Fore.RED+"неверно! Введите число!"+Style.RESET_ALL)
            self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
            return
        sql = "DELETE FROM " + self.table_name()
        sql += f" WHERE {self.primary_key()[0]} = (%s)"
        #print(sql)
        cur = self.dbconn.conn.cursor()
        cur.execute(sql,(str(id_),))
        self.dbconn.conn.commit()


    def __call_creation_wizard(self)->list:
        rid = None
        max_spaces = None
        l=w=h = 1
        max_weight = 0
        rt = RoomTable()
        """
        Мастер создания полок
        """
        while not(type(rid) == int and rid in rt.create_list_of_ids() and (0 < rid
<= pglimits.PG_INT_MAX)):
            try:
                rid = int(input(Fore.YELLOW+"В какую комнату добавить новую полку?
для отмены введите 0: "+Style.RESET_ALL))
                if not rid:
                    return
                if ((rid not in rt.create_list_of_ids()) or not(pglimits.PG_INT_MIN
```

```
<= rid <= pglimits.PG_INT_MAX)):
                    raise ValueError
            except ValueError as e:
                print(Fore.RED+"ошибка, введите верное число"+Style.RESET_ALL)
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
        while not (type(max_spaces) == int and (0 < max_spaces <=
pglimits.PG_INT_MAX)):
            try:
                max_spaces = int(input(Fore.YELLOW+"Введите количество мест на
полке для отмены введите 0: "+Style.RESET_ALL))
                if not max_spaces:
                    return
                if not(0 < max_spaces <= pglimits.PG_INT_MAX):
                    raise ValueError
            except ValueError as e:
                print(Fore.RED+"Введите число, то что ты ввел, редиска, не
число"+Style.RESET_ALL)
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
            if max_spaces < 0:
                print(Fore.RED+"Недопустимое количество мест")
                self.dbconn.logger.warn(Fore.GREEN+str("Недопустимое количество
мест")+Style.RESET_ALL)
        spaces_left = max_spaces
        while not ((type(l) == float) and ((0<l<=pglimits.NUMERIC7_0_MAX))
            and (type(w) == float) and ((0<w<=pglimits.NUMERIC7_0_MAX))
            and (type(h) == float) and ((0<h<=pglimits.NUMERIC7_0_MAX))):

            try:
                l,w,h = map(float, input(Fore.YELLOW+"Введите габариты места на
полке, разделяя их пробелом: "+Style.RESET_ALL).split())
                if not((pglimits.NUMERIC7_0_MIN<=l<=pglimits.NUMERIC7_0_MAX) or
(pglimits.NUMERIC7_0_MIN<=h<=pglimits.NUMERIC7_0_MAX) or (pglimits.NUMERIC7_0_MIN<=
w<=pglimits.NUMERIC7_0_MAX)):
                    raise ValueError
            except ValueError as e:
                print(Fore.RED+"ты в курсе что такое три числа?"+Style.RESET_ALL)
                print(e)
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
            if(l <= 0 or w <= 0 or h <= 0):
                print(Fore.RED+"Недопустимый габарит")

        while not((type(max_weight)) and (0 < max_weight <=
pglimits.NUMERIC7_2_MAX)):
            try:
                max_weight = float(input(Fore.YELLOW+"Введите максимальный
нагрузочный вес полки для отмены введите 0: "+Style.RESET_ALL))
```

```
                if not max_weight:
                    return
                if not(0 < max_weight <= pglimits.NUMERIC7_2_MAX):
                    raise ValueError
            except ValueError as e:
                print(Fore.RED+"Вес - число, а не то что ты ввел"+Style.RESET_ALL)
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
        if max_weight <= 0:
            print(Fore.RED+"Недопустимый вес"+Style.RESET_ALL)
        weight_left = max_weight
        return [rid,max_spaces,spaces_left,w,h,l,max_weight,weight_left]




    def show_shelves(self)->None:
        """
        отобразить полки
        """
        menu = Fore.YELLOW+""" Просмотр списка полок
№\tКомната\tмакс мест\tоставшиеся места\tгабариты места\tвес макс\t вес
оставшийся""" + Style.RESET_ALL
        print(menu)
        lst = self.all()
        for i in lst:
            print(str(i[0]) + "\t" + str(i[1]) + "\t" + str(i[2]) + "\t" +
str(i[3]) + "\t"
                f"{str(i[4])} x {str(i[5])} x {str(i[6])}" + "\t" + str(i[7]) +
"\t" + str(i[8]))


    def add_shelf_attached_to_room(self)-> None:
        """
        обработчик создания полки
        """
        data = self.__call_creation_wizard()
        if type(data) == list:
            self.insert_one(data)
            print(Fore.GREEN + "полка создана"+ Style.RESET_ALL)

    def del_shelf_by_room(self,room_id:int):
        """
        удаление полки
        """
        lst = self.all_by_room_id(room_id)
        for i in lst:
```

```
            print(str(i[0]) + "\t" + str(i[1]) + "\t" + str(i[2]))
            self.delete_shelf()
            self.all_by_room_id(room_id)




    def edit_check(self,id_:int, col_2edit: int, new_data:str)-> bool:
        """
        проверка выполнения ограничений целостности при редактировании полки
        """
        if (col_2edit == 0):
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            try:
                RT = RoomTable()
                ids = RT.create_list_of_ids()
                if(int(new_data) not in ids):
                    raise ValueError
            except ValueError as e:
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
                print(Fore.RED +"Введено неверное значение" + Style.RESET_ALL)
                new_data = input(Fore.YELLOW + "Введите заново: "+ Style.RESET_ALL)
                return self.edit_check(id_,col_2edit,new_data)
            cur.execute(sql, [new_data, str(id_)])
            self.dbconn.conn.commit()
            return True
        elif (col_2edit == 1):
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            try:
                sql_ = f"SELECT {self.column_names_without_id()[col_2edit+1]} FROM
{self.table_name()} WHERE {self.primary_key()[0]} = (%s)"
                cur = self.dbconn.conn.cursor()
                cur.execute(sql_, tuple(str(id_),))
                recived = list(cur.fetchone())[0]

                if not(float(new_data) >= recived):
                    raise ValueError
            except ValueError as e:
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
```

```
                print(Fore.RED +"Введено неверное значение" + Style.RESET_ALL)
                new_data = input(Fore.YELLOW + "Введите заново: "+ Style.RESET_ALL)
                return self.edit_check(id_,col_2edit,new_data)
            cur.execute(sql, [new_data, str(id_)])
            self.dbconn.conn.commit()
            return True
        elif (col_2edit == 3):
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            try:
                if (int(new_data) <= 0):
                    raise ValueError
            except ValueError as e:
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
                print(Fore.RED +"Введено неверное значение" + Style.RESET_ALL)
                new_data = input(Fore.YELLOW + "Введите заново: "+ Style.RESET_ALL)
                return self.edit_check(id_,col_2edit,new_data)
            cur.execute(sql, [new_data, str(id_)])
            self.dbconn.conn.commit()
            return True
        elif (col_2edit == 4):
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            try:
                if (int(new_data) <= 0):
                    raise ValueError
            except ValueError as e:
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
                print(Fore.RED +"Введено неверное значение" + Style.RESET_ALL)
                new_data = input(Fore.YELLOW + "Введите заново: "+ Style.RESET_ALL)
                return self.edit_check(id_,col_2edit,new_data)
            cur.execute(sql, [new_data, str(id_)])
            self.dbconn.conn.commit()
            return True
        elif (col_2edit == 5):
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            try:
                if (int(new_data) <= 0):
                    raise ValueError
```

```
            except ValueError as e:
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
                print(Fore.RED +"Введено неверное значение" + Style.RESET_ALL)
                new_data = input(Fore.YELLOW + "Введите заново: "+ Style.RESET_ALL)
                return self.edit_check(id_,col_2edit,new_data)
            cur.execute(sql, [new_data, str(id_)])
            self.dbconn.conn.commit()
            return True
        elif (col_2edit == 6):
            sql = "UPDATE " + self.table_name() + " SET "
+self.column_names_without_id()[col_2edit]
            sql += " = (%s) WHERE " + self.primary_key()[0] + " = (%s)"
            cur = self.dbconn.conn.cursor()
            try:
                sql_ = f"SELECT {self.column_names_without_id()[col_2edit+1]} FROM
{self.table_name()} WHERE {self.primary_key()[0]} = (%s)"
                cur = self.dbconn.conn.cursor()
                cur.execute(sql_, (str(id_),))
                recived = list(cur.fetchone())[0]

                if not(float(new_data) >= recived):
                    raise ValueError
            except ValueError as e:
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
                print(Fore.RED +"Введено неверное значение" + Style.RESET_ALL)
                new_data = input(Fore.YELLOW + "Введите заново: "+ Style.RESET_ALL)
                return self.edit_check(id_,col_2edit,new_data)
            cur.execute(sql, [new_data, str(id_)])
            self.dbconn.conn.commit()
            return True




    def edit_shelf(self):
        """
        редактор полок
        """
        inp = None
        col_2edit = None

        self.show_shelves()
        while not (inp in self.create_list_of_ids()):
            try:
                inp = int(input(Fore.YELLOW + "Выберите полку которую хотите
изменить для отмены введите 0: " + Style.RESET_ALL))
```

```
                if not inp:
                    return
                if inp not in self.create_list_of_ids():
                    raise ValueError
            except ValueError as e:
                print(Fore.RED+ "Введите правильное число" + Style.RESET_ALL)
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)

        sql = f"SELECT * FROM {self.table_name()} WHERE {self.primary_key()[0]} =
(%s)"
        cur = self.dbconn.conn.cursor()
        cur.execute(sql, (str(inp),))
        recived = list(cur.fetchone())[1:]
        cols = self.column_names_without_id()
        data = list(zip(cols,recived))
        for col in enumerate(data):
            print(col[0],col[1], sep = "\t")
        while not(type(col_2edit) == int and (0 <= col_2edit < len(data))):
            try:
                col_2edit = int(input(Fore.YELLOW + "Введите номер поля, который
хотите изменить: для отмены введите -1 " + Style.RESET_ALL))
                if(col_2edit == -1):
                    return
                if not(0 <= col_2edit < len(data)):
                    raise ValueError
            except ValueError as e:
                self.dbconn.logger.warn(Fore.GREEN+str(e)+Style.RESET_ALL)
                print(Fore.RED + "Введено неверное число"+Style.RESET_ALL)
        new_data = input(Fore.YELLOW+"Введите новое значение поля:
"+Style.RESET_ALL)
        if self.edit_check(inp,col_2edit,new_data):
            print(Fore.GREEN+"Изменения применены"+Style.RESET_ALL)
            return
        print(Fore.RED+"Во внесении изменений отказно, неверное новое
значение"+Style.RESET_ALL)
        return
```

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

```python
from project_config import *
from dbconnection import *
from dbtable import DbTable
from room_table import RoomTable
from shelf_table import ShelfTable
import psycopg2.errors
import colorama
from colorama import Fore, Back, Style

colorama.init()

class Main:

    config = ProjectConfig()
    connection = DbConnection(config)

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

    def db_init(self):
        """
        инициализация таблиц
        """
        rt = RoomTable()
        st = ShelfTable()
        st.drop()
        rt.drop()
        rt.create()
        st.create()
        return

    def db_insert_somethings(self):
        """
        заполнение некими данными
        """
        rt = RoomTable()
        st = ShelfTable()
        rt.insert_one(['Room1',50,50,20,60,18,32])
        rt.insert_one(['Room2',500,500,40,100,22,32])
        rt.insert_one(['Room3',50,50,20,60,18,32])
        rt.insert_one(['Garage1',10000,10000,40,100,22,32])
        rt.insert_one(['Room4',50,50,20,60,18,32])

        st.insert_one([1,10,10,500,400,300,500,500])
        st.insert_one([1,50,50,500,400,400,600,600])
```

```
        st.insert_one([2,40,40,500,500,500,500,500])
        st.insert_one([3,40,40,500,500,500,500,500])
        st.insert_one([4,5,5,500,500,500,40,40])

    def db_drop(self):
        """
        drop структуры
        """
        rt = RoomTable()
        st = ShelfTable()
        st.drop()
        rt.drop()


    def show_main_menu(self):
        menu = Fore.YELLOW +"""Дальнейшие операции:
    """+Style.RESET_ALL +Fore.GREEN + str(1) + Style.RESET_ALL+"""  -
взаимодействовать с комнатами
    """+Fore.GREEN + str(2) + Style.RESET_ALL+"""  - Очистка и создание таблиц
    """+Fore.GREEN + str(3) + Style.RESET_ALL+"""  - Взаимодействовать с полками
    """+Fore.GREEN + str(9) + Style.RESET_ALL+"""  - выход."""
        print(menu)
        return

    def read_next_step(self):
        """
        отобразить приглашение ввода
        """
        return input(Fore.YELLOW +"=> " + Style.RESET_ALL).strip()

    def after_main_menu(self, next_step):
        """
        переход из главного меню
        """
        if next_step == "2":
            self.db_drop()
            self.db_init()
            self.db_insert_somethings()
            print(Fore.GREEN + "Таблицы созданы заново!"+ Style.RESET_ALL)
            return "0"
        elif next_step != "1" and next_step != "9" and next_step != "3":
            print(Fore.RED + "Выбрано неверное число! Повторите ввод!" +
Style.RESET_ALL)

            return "0"
        else:
```

```
            return next_step
    def after_show_people(self, next_step):
        """
        меню комнат, обработка перехода
        """

        while True:
            if next_step == "4":
                RT = RoomTable()
                RT.delete_room()
                return "1"
            elif next_step == "7":
                print("Пока не реализовано!")
            elif next_step == '6':
                RT = RoomTable()
                RT.edit_room()
                next_step = "0"
            elif next_step == "3":
                RT = RoomTable()
                RT.add_rooms()
                next_step = "0"
            elif next_step == "5":
                ST = ShelfTable()
                RT = RoomTable()
                RT.show_rooms()
                rid = int(input(Fore.YELLOW +"выберите комнату для просмотра полок:
" + Style.RESET_ALL))
                data = ST.all_by_room_id(rid)
                print(data)
                next_step = "0"
            elif next_step != "0" and next_step != "9" and next_step != "3":
                print(Fore.RED + "Выбрано неверное число! Повторите ввод!" +
Style.RESET_ALL)
                return "1"
            else:
                return next_step
    def display_shelves_menu(self):
        """
         Меню полки + обработчик перехода
         """

        menu = Fore.YELLOW +"""Дальнейшие операции:
    """+Style.RESET_ALL +Fore.GREEN + str(0) + Style.RESET_ALL+"""  - возврта в
главное меню
    """+Fore.GREEN + str(3) + Style.RESET_ALL+"""  - добавление новой полки к
комнате;
```

```
    """+Fore.GREEN + str(4) + Style.RESET_ALL+"""  - удаление полки;
    """+Fore.GREEN + str(5) + Style.RESET_ALL+"""  - просмотр стеллажей комнаты;
    """+Fore.GREEN + str(6) + Style.RESET_ALL+"""  - редактирование полки
    """+Fore.GREEN + str(9) + Style.RESET_ALL + "  - выход."""

        print(menu)


        ST = ShelfTable()
        user_chose = input(Fore.YELLOW +"выберите нужный пункт меню: " +
Style.RESET_ALL)
        if user_chose == "0":
            return
        elif user_chose == "3":
            ST.add_shelf_attached_to_room()
            return
        elif user_chose == '4':
            ST.delete_shelf()
            ST.show_shelves()
            return
        elif user_chose == "5":
            RT = RoomTable()
            RT.show_rooms()
            rid = int(input(Fore.YELLOW +"выберите комнату для просмотра полок: " +
Style.RESET_ALL))
            data = ST.all_by_room_id(rid)
            print(data)
            return
        elif user_chose == "6":
            ST.edit_shelf()
            return
    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":
                RT = RoomTable()
                RT.show_rooms()
                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.display_shelves_menu()
                current_menu = "0"
        print(Fore.CYAN + "До свидания!"+Style.RESET_ALL)
        return



    def test(self):
        DbTable.dbconn.test()

m = Main()
try:
    m.main_cycle()
except psycopg2.errors.UndefinedTable as UndefinedTable:
    print(Fore.RED +"Кажется заданная таблица не найдена, проверьте структуру базы
данных или выполните действие 2 из главного меню, чтобы создать Таблицы\n" +
Style.RESET_ALL
        , UndefinedTable)
except psycopg2.errors.CheckViolation:
    print(Fore.RED+"Нарушение ограничений целостности" + Style.RESET_ALL)
    m.main_cycle()
except Exception as e:
    print(Fore.RED+"Что-то пошло не так"+Style.RESET_ALL)
    cur = m.connection.conn.cursor()
    cur.execute("ROLLBACK TRANSACTION;")
    try:
        connection.logger.warn(e)
    except Exception as e:
        print(Fore.RED+"лог файл недоступен"+Style.RESET_ALL)
    m.main_cycle()#
```