

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Лабораторна робота №2**

з дисципліни **Бази даних і засоби управління**

*на тему: “Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL”*

Виконав:

студент ІII курсу

групи КВ-91

Селетков В. Р.

Перевірив:

Павловський В. І.

Київ – 2021

# Постановка задачі

*Метою роботи* є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

*Загальне завдання* роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових, – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання- контролер).

**Інформація про програму**

Посилання на репозиторій у GitHub з вихідним кодом програми та прикладеним звітом: <https://github.com/vladsel/database>

Використана мова програмування: Python 3.10.

Використані бібліотеки: psycopg2 (для зв’язку з СУБД), time (для виміру часу запиту пошуку, що у 3 завданні)

Використаний шаблон проектування: MVC.

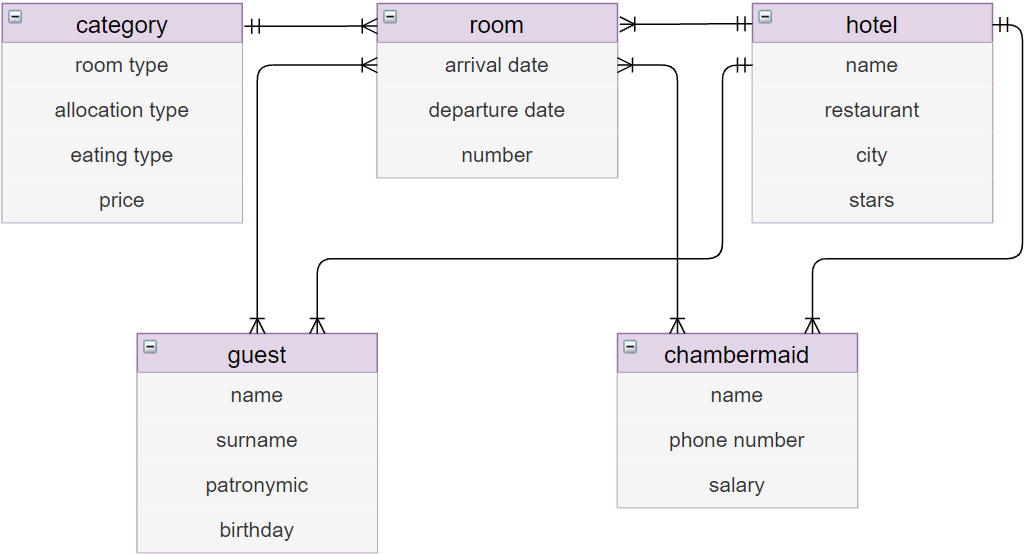
**Відомості про предметну галузь з лабораторної роботи №1**

Рисунок 1 - ER-діаграма побудована за нотацією “Пташиної лапки (Crow’s foot)”, задана ER-діаграма була побудована у додатку [draw.io](https://app.diagrams.net/)

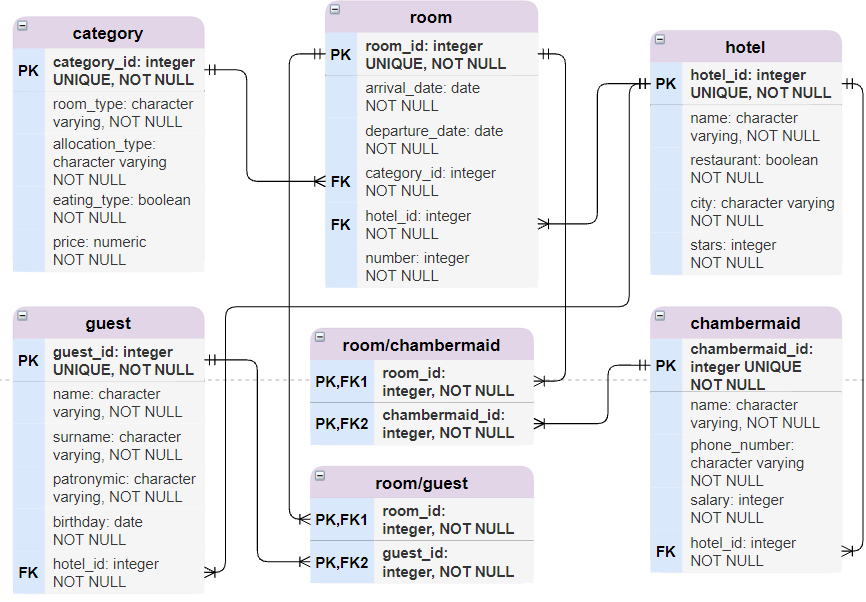
****

Рисунок 2 - Схема бази даних, побудовано у додатку [draw.io](https://app.diagrams.net/)

**Таблиця 1 - Опис структури БД.**

|  |  |  |
| --- | --- | --- |
| Відношення | Атрибут | Тип атрибуту |
| **hotel –** містить дані про готель | **hotel\_id –** унікальний ідентифікатор  **name –** назва готелю  **restaurant –** наявність ресторану  **city –** місто  **stars –** кількість зірок | **integer** (числовий)  **character varying** (рядок)  **boolean** (булевий)  **character varying** (рядок)  **integer** (числовий) |
| **category –** містить дані про категорію номеру у готелі | **category\_id** – унікальний ідентифікатор  **room\_type –** тип номеру  **allocation\_type –** тип розселення в номері  **eating\_type –** наявність харчування  **price –** ціна | **integer** (числовий)  **character varying** (рядок)  **character varying** (рядок)  **boolean** (булевий)  **numeric** (фіксований) |
| **guest –** містить дані про постояльців готелю | **guest\_id** – унікальний ідентифікатор  **name** – ім’я  **surname –** прізвище  **patronymic –** по батькові  **birthday –** день народження  **hotel\_id –** ідентифікатор готелю | **integer** (числовий)  **character varying** (рядок)  **character varying** (рядок)  **character varying** (рядок)  **date** (дата)  **integer** (числовий) |
| **room** – містить дані щодо номеру | **room\_id -** унікальний ідентифікатор  **arrival\_date –** дата заселення  **departure\_date –** дата виселення  **category\_id** – ідентифікатор категорії  **hotel\_id –** ідентифікатор готелю  **number –** номер кімнати | **integer** (числовий)  **date** (дата)  **date** (дата)  **integer** (числовий)  **integer** (числовий)  **integer** (числовий) |
| **chambermaid –** містить дані про покоївок готелю | **chambermaid\_id –** унікальний ідентифікатор  **name –** ім’я  **phone –** номер телефону  **salary –** заробітня плата  **hotel\_id –** ідентифікатор готелю | **integer** (числовий)  **character varying** (рядок)  **character varying** (рядок)  **integer** (числовий)  **integer** (числовий) |
| **room/**  **chambermaid** -  відношення покоївок до кімнат | **room\_id–** ідентифікатор номера  **chambermaid\_id –** ідентифікатор покоївки | **integer** (числовий)  **integer** (числовий) |
| **room/guest** -  відношення постояльців до кімнат | **room\_id–** ідентифікатор номера  **chambermaid\_id –** ідентифікатор постояльця | **integer** (числовий)  **integer** (числовий) |

У Обраній базі даних «Готель» можна виділити наступні таблиці: загальні відомості про готель (hotel), тип заданого номера (room), категорія номера (category), загальні відомості про постояльця (guest), інформація про покоївку (chambermaid), відношення покоївок до кімнат (room/chambermaid), відношення постояльців до кімнат (room/guest).

Стовпці заданих таблиць:

* + - 1. hotel: hotel\_id, name, restaurant, city, stars.
      2. room: room\_id, arrival date, departure date, category\_id, hotel\_id, number.
      3. category: category\_id, room type, allocation type, eating type, price.
      4. guest: guest\_id, name, surname, patronymic, birthday, hotel\_id.
      5. chambermaid: chambermaid\_id, name, phone number, salary, hotel\_id.

**Структура програми**

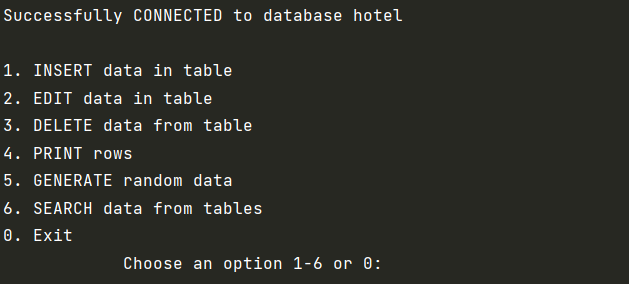
За шаблоном проектування MVC, додаток складається з таких модулей:

*model.py* — підключається до БД та виконує операції – SELECT, INSERT, DELETE, UPDATE та, більш складні операції з БД.

*controller.py* — головне та допоміжні меню, для зручного керування БД.

*mainl.py* — точка входу в програму, та підключення до БД.

# Схема меню користувача



На знімку екрану термінала продемонстровано початкове меню, де можна побачити функції які можна виконати з БД. Кожна команда запускає відповідну функцію з файлу controller.py, яка в подальшому передає аргументи у функцію файлу model.py, яка в свою чергу формує і здійснює запит до бази даних.

***Методи реалізовані до пункту 1 завдання лабораторної роботи:***

1. INSERT data in table — викликає функцію вставки даних у таблицю бази даних;
2. EDIT data in table — викликає функцію редагування даних у таблиці бази даних;
3. DELETE data from table — викликає функцію видалення даних у таблиці бази даних;
4. PRINT rows — викликає функцію виводу даних з таблиці бази даних.

***Метод реалізований до пункту 2 завдання лабораторної роботи:***

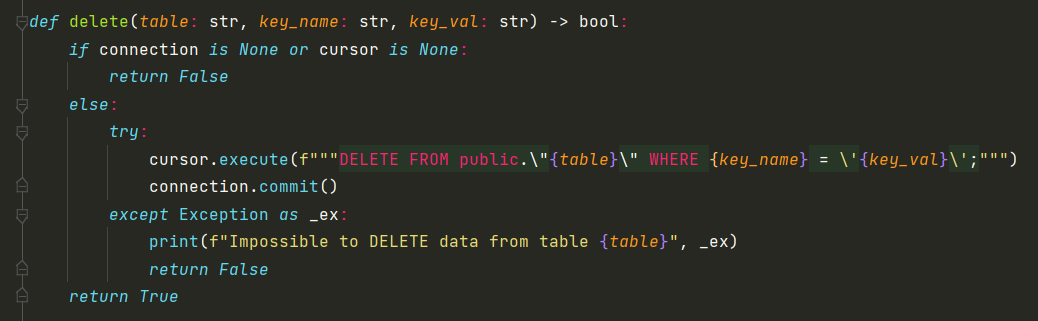
1. GENERATE random data — дозволяє користувачеві заповнити таблицю, або всю базу даних випадково згенерованими даними.

Кожна з описаних вище функцій викликає допоміжну функцію select\_table(), даний скрипт дозволяє користувачеві обирати таблицю

***Метод реалізований до пункту 3 завдання лабораторної роботи:***

1. SEARCH data from tables — викликає функцію пошуку даних у таблицях за атрибутами та поєднання таблиць за ключем.

**Завдання 1**

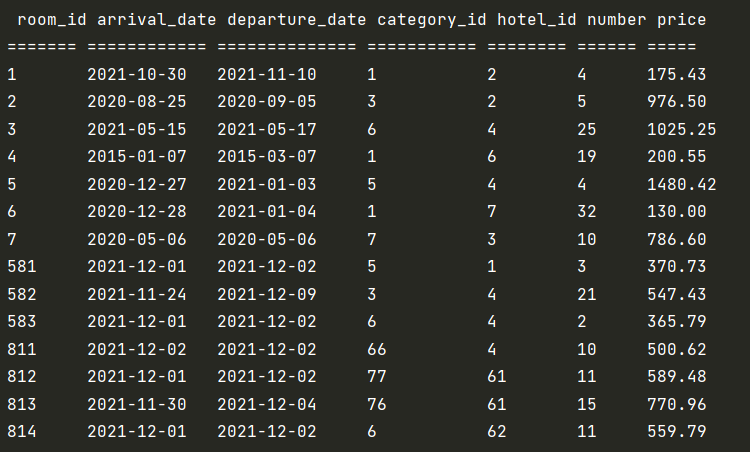
***Запит на видалення***

Для перевірки роботи розглянемо запити на видалення даних з дочірньої таблиці **room** та батьківської таблиці **hotel**.

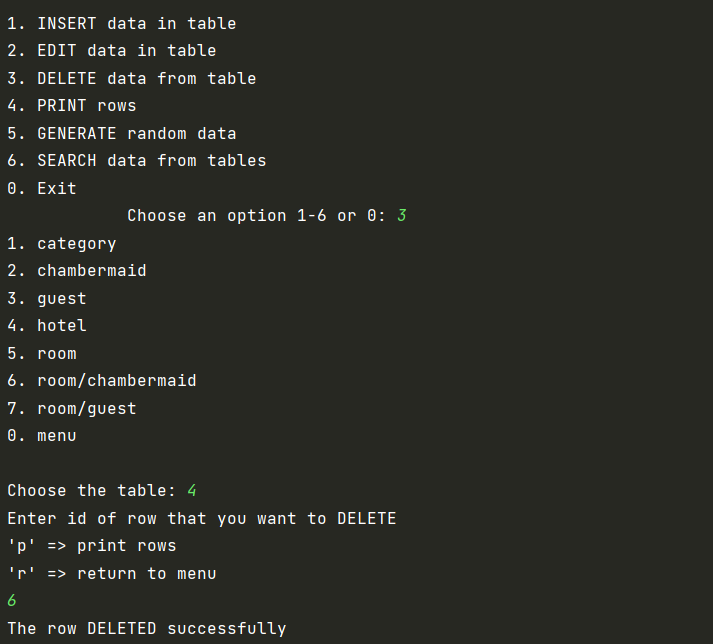
Таблиця **hotel** до видалення даних:



Таблиця **room** до видалення даних:



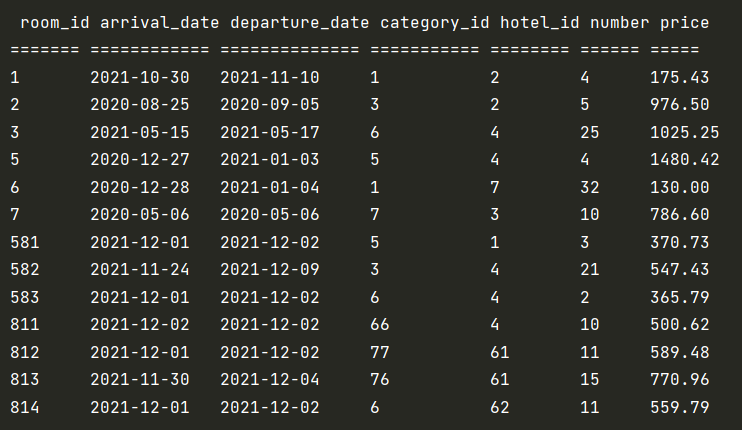
У даній програмній реалізації видалення запису з батьківської таблиці, який зв’язаний з дочірньою таблицею, буде видалено каскадно, тобто всі дані з цим зовнішнім ключем будуть видалені.

Видалимо з таблиці **hotel** рядок з hotel\_id = 6:

Таблиця **hotel** після видалення даних:

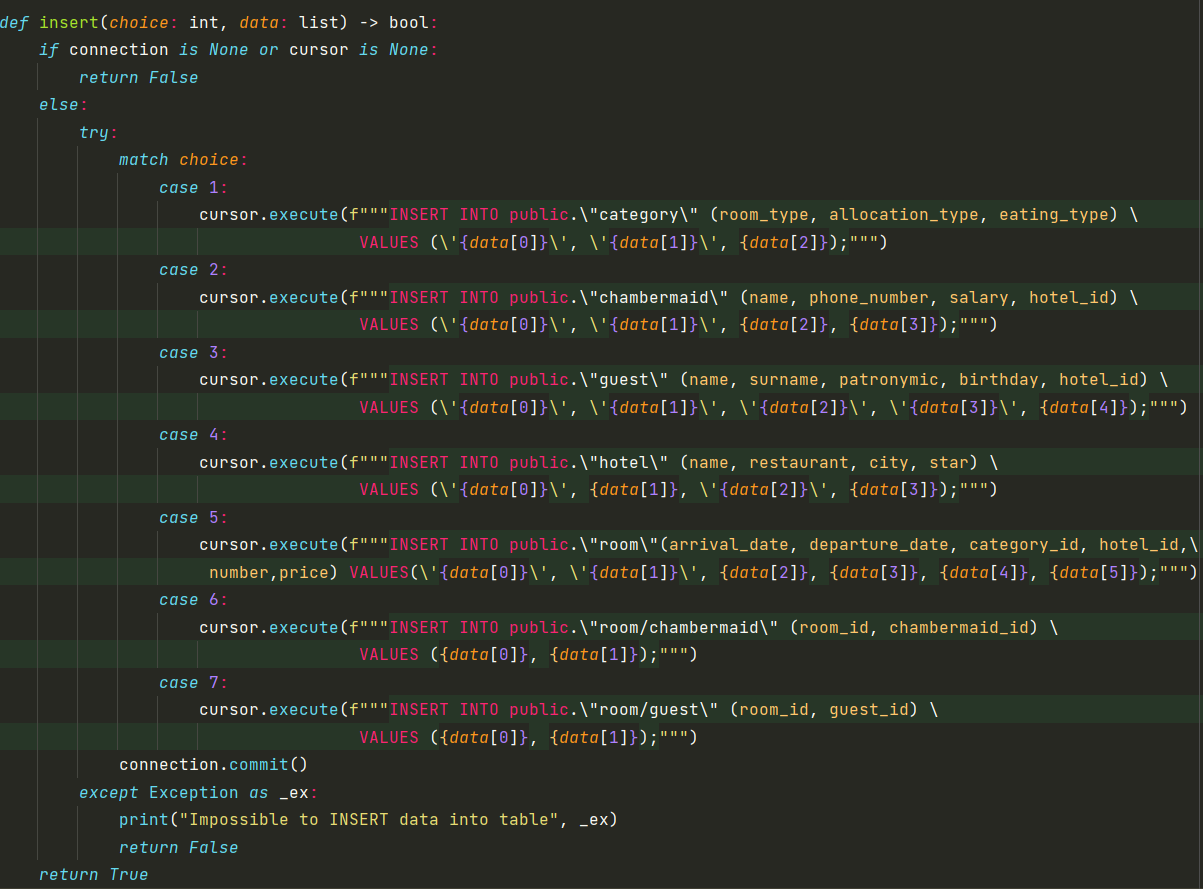


Таблиця **room** після видалення даних:

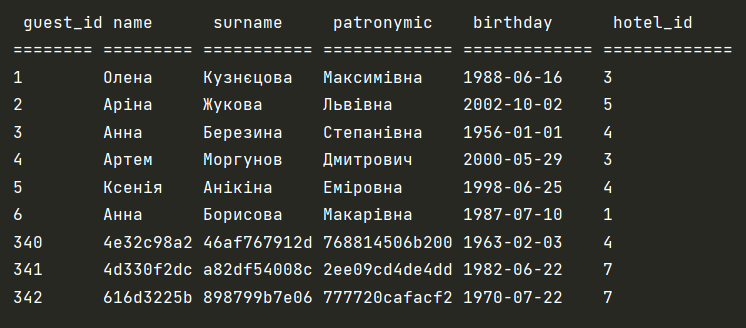


З даних екранних відображень можна помітити, що при видаленні з таблиці **hotel** рядку з hotel\_id = 6, також було видалено рядок з таблиці **room** зі значеннями, де room\_id = 4 та hotel\_id = 6.

***Запит на вставку поля***



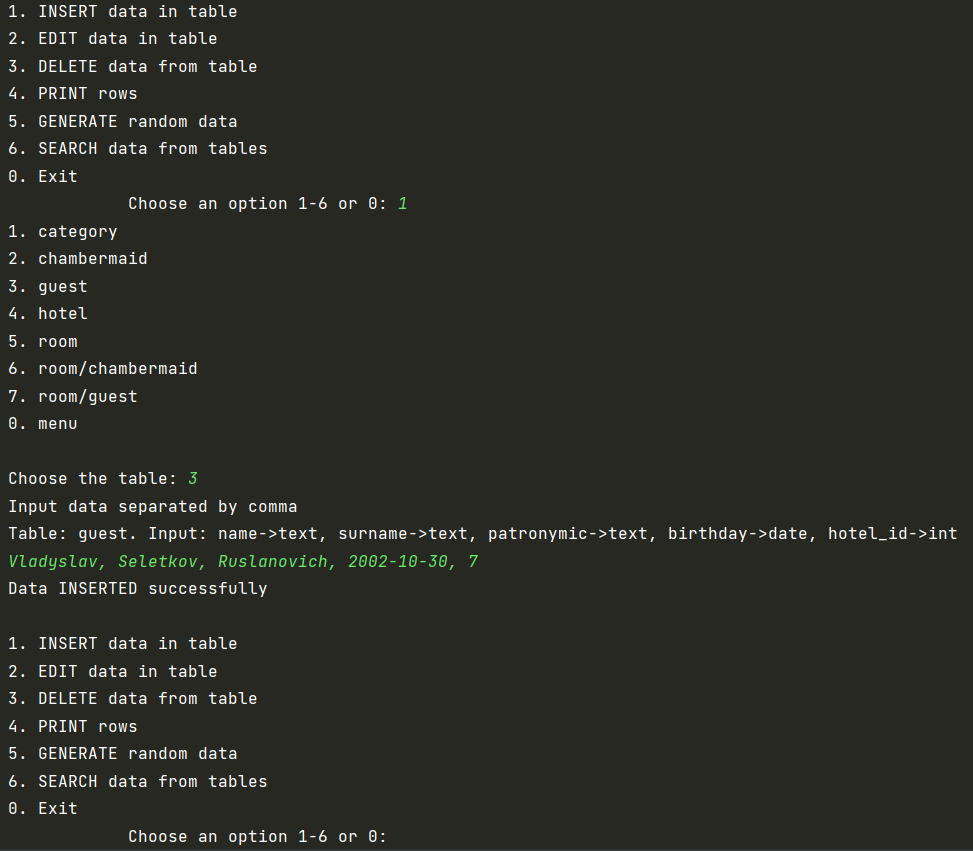
Для перевірки роботи розглянемо запити на вставку в таблиці **guest**. Спочатку спробуємо коректний запис, а потім з неіснуючим значенням зовнішнього ключа таблиці **hotel**.

Таблиця **guest** до вставки даних:

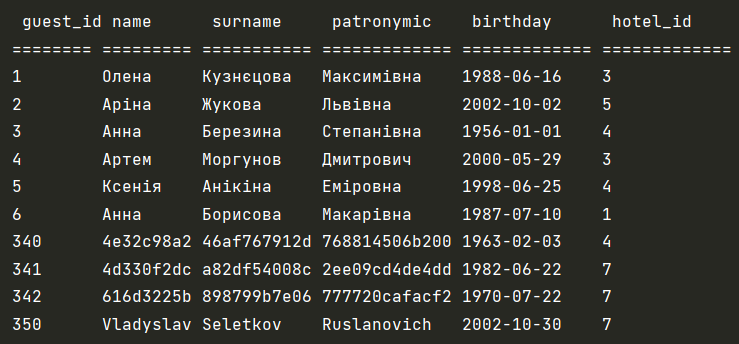
Таблиця **hotel** до вставки даних:



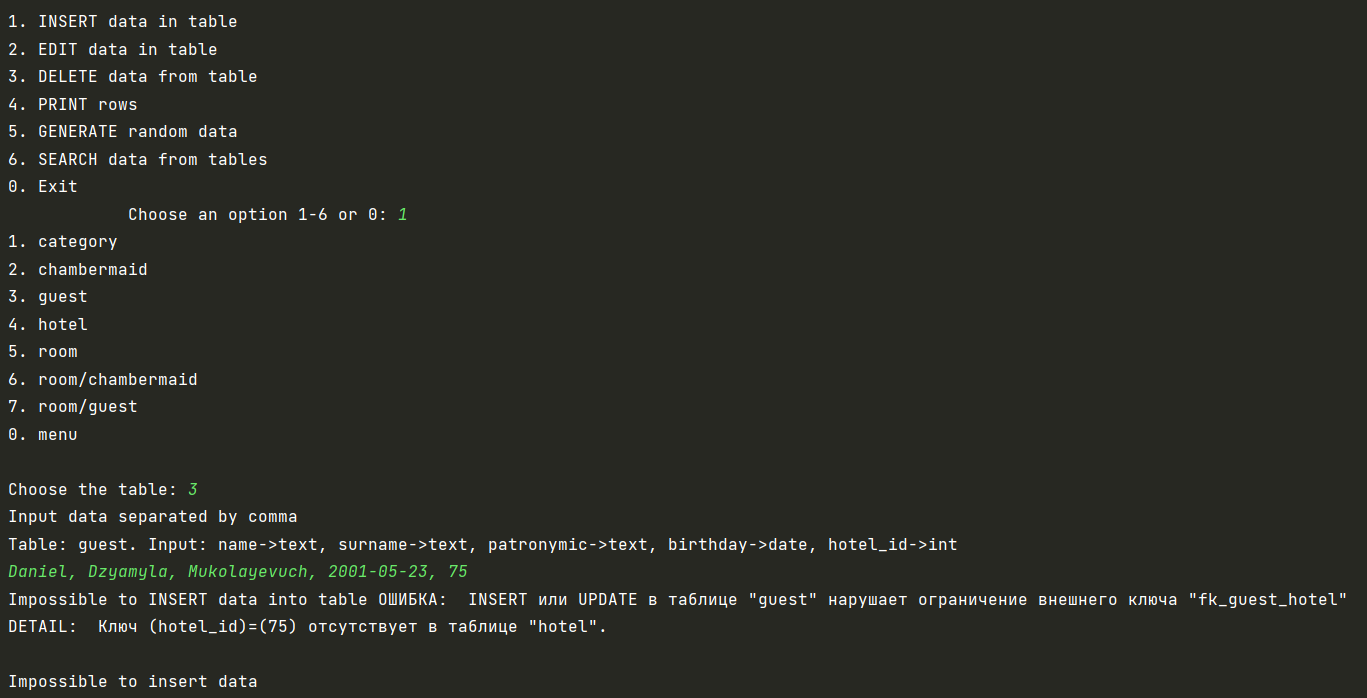
Вставимо в таблицю **guest** нові дані (guest\_id = 350):



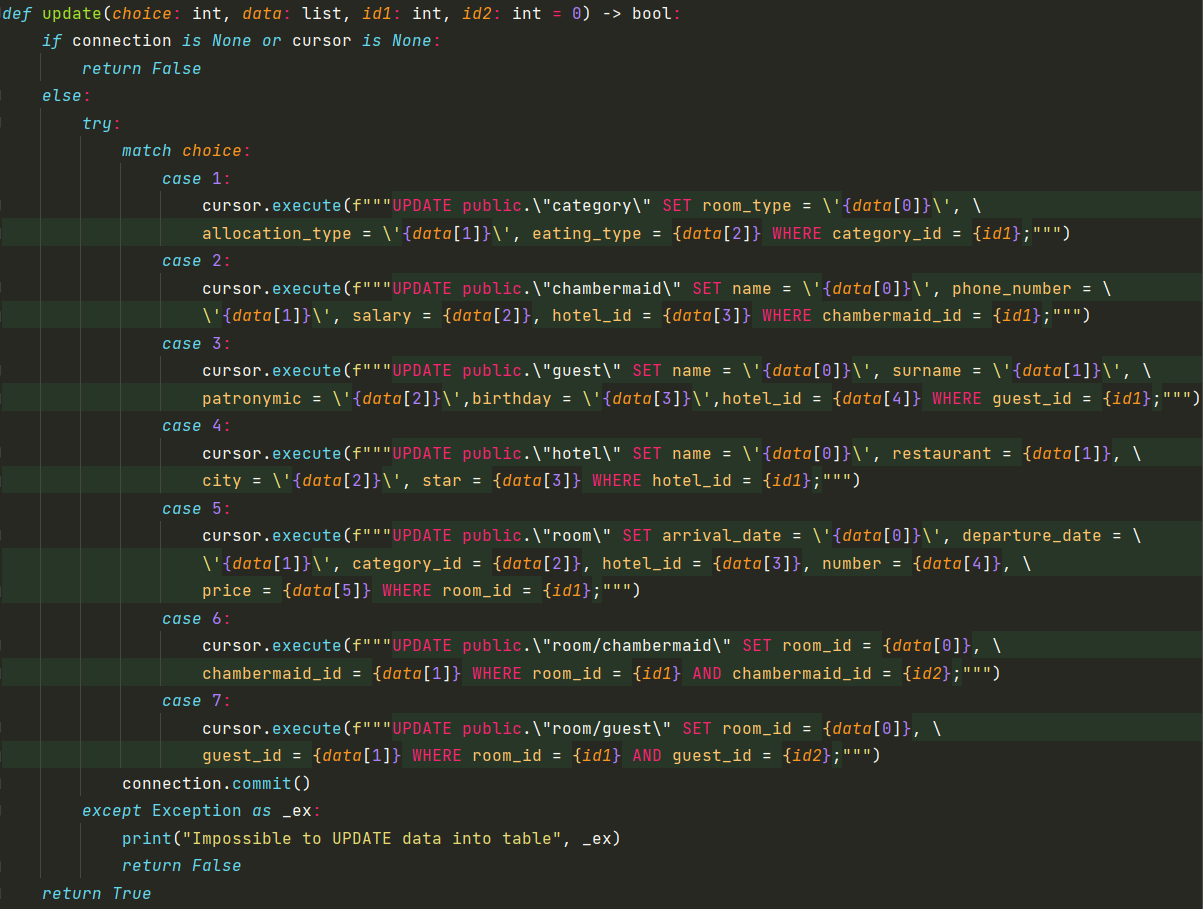
Таблиця **guest** після вставки даних (guest\_id = 350):



Можна помітити, що вставка виконалася правильно, оскільки зовнішній ключ hotel\_id посилається на існуючий запис у табличці **hotel**.

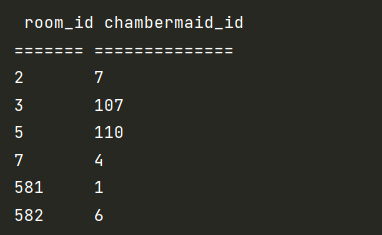
 Виконаємо вставку в таблицю **guest** з неіснуючим значенням hotel\_id:

На екранному відображенні видно, що виникає помилка, оскільки зовнішній ключ hotel\_id посилається на неіснуючий запис у табличці **hotel**.

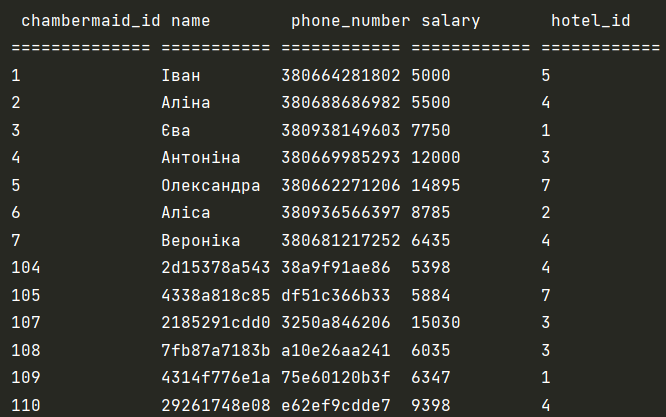
***Запит на редагування рядків***

Для перевірки роботи розглянемо запити на редагування в таблиці **room/chambermaid**. Спочатку спробуємо коректний запис, а потім з неіснуючим значенням зовнішнього ключа таблиці **chambermaid**.

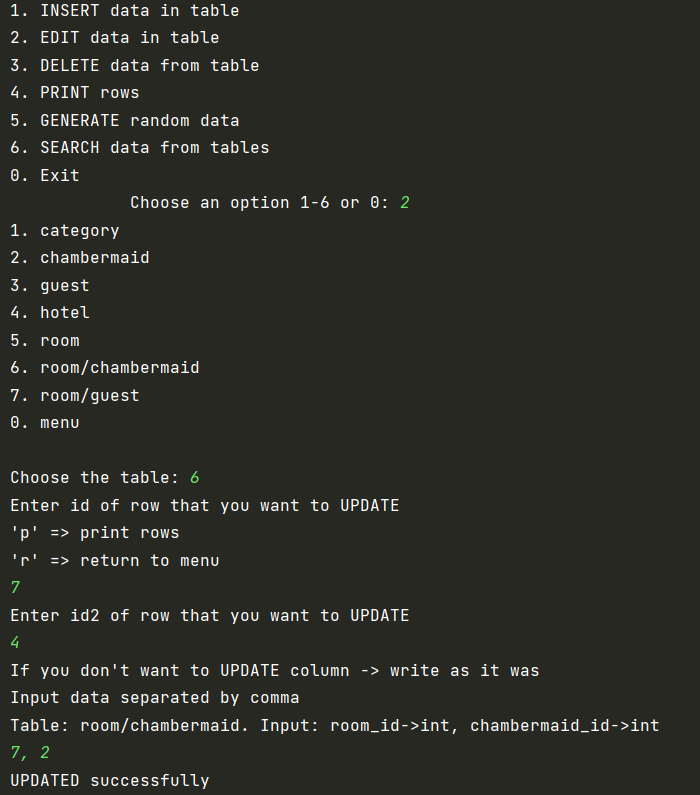
Таблиця **room/chambermaid** до редагування даних:



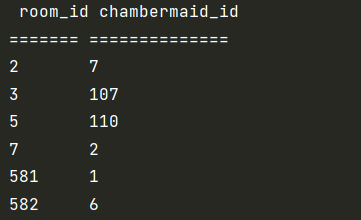
Таблиця **chambermaid** до редагування даних:



Відредагуємо в таблиці **room/chambermaid** рядок з room\_id = 7 та chambermaid\_id = 4:

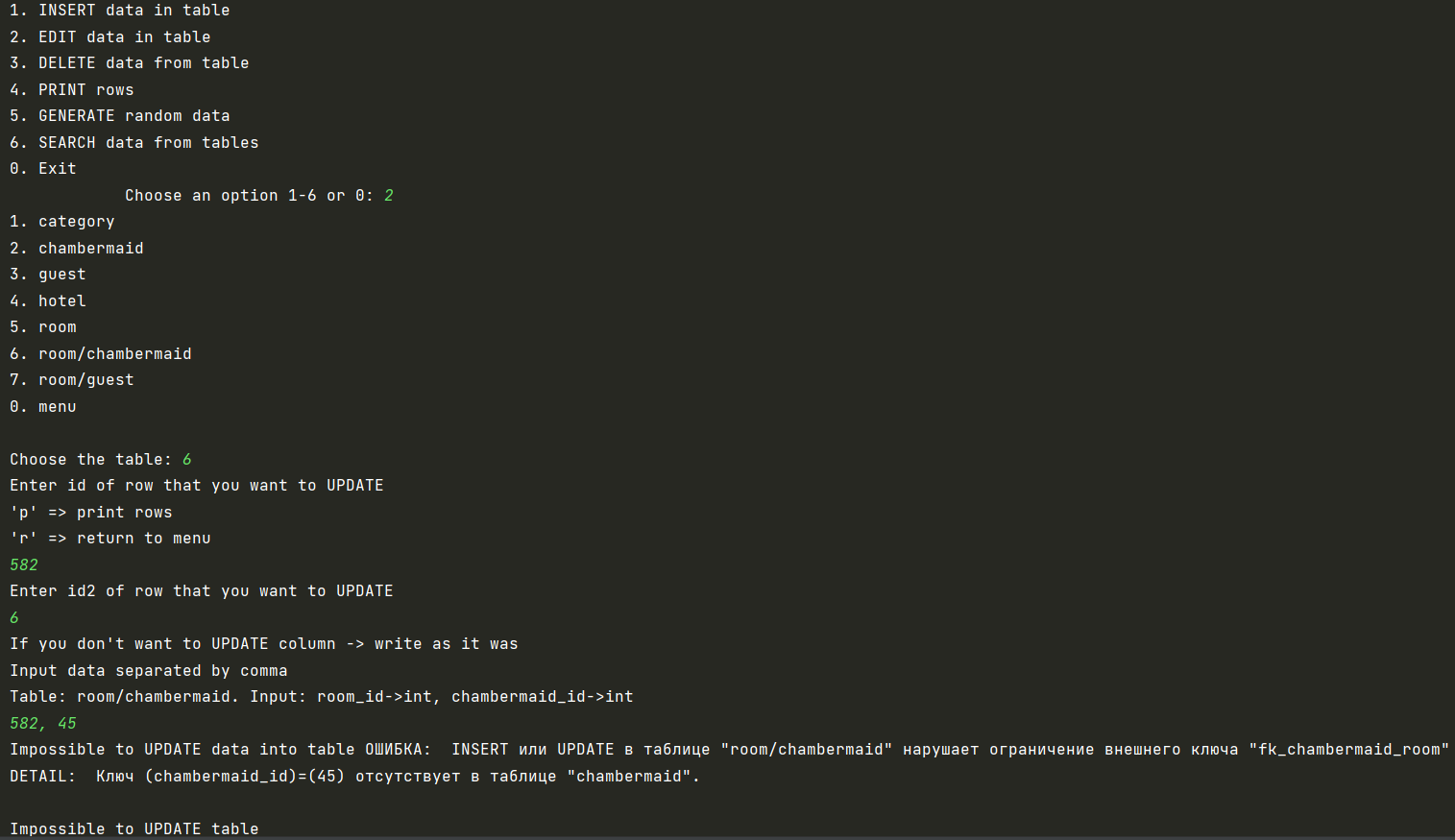


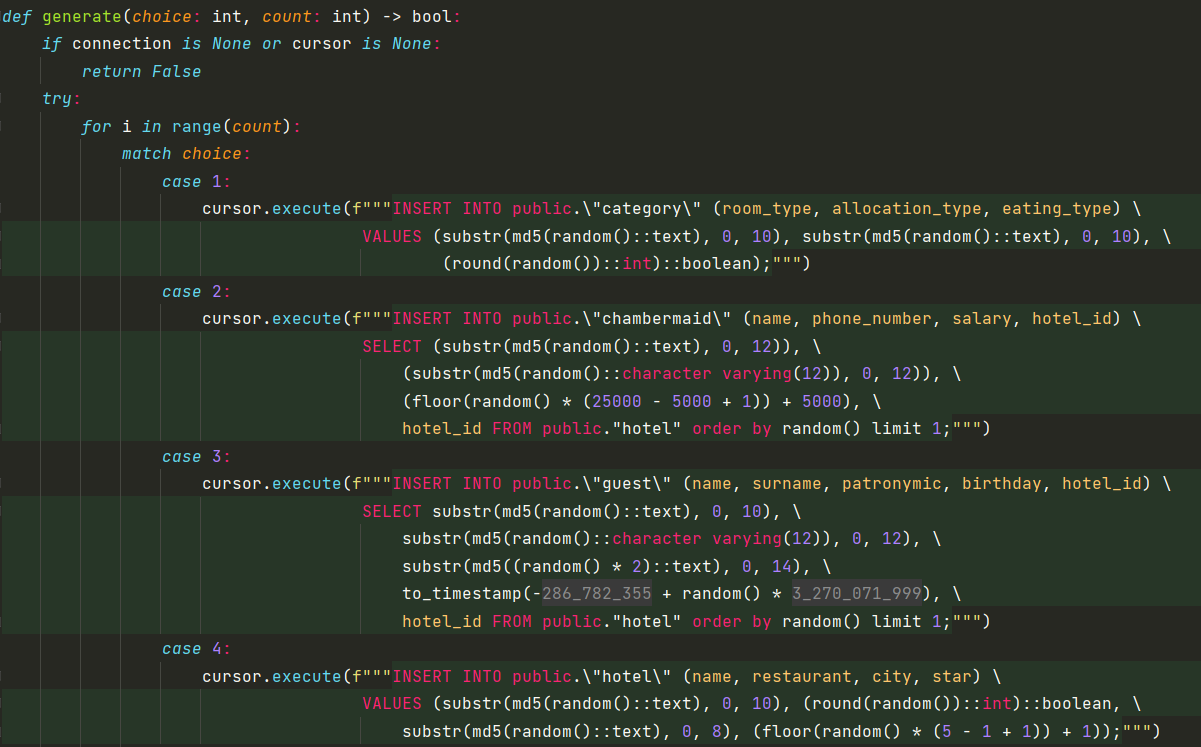
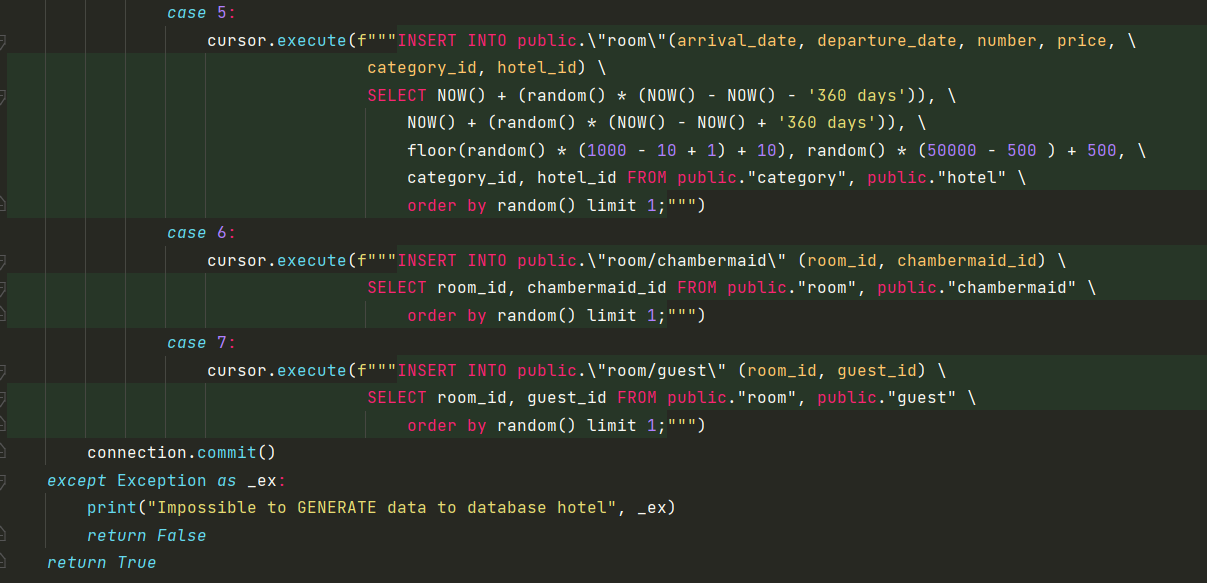
Таблиця **room/chambermaid** після редагування даних:



Можна помітити, що редагування виконалося правильно (room\_id = 7 та chambermaid\_id = 2), зовнішній ключ chambermaid\_id посилається на вже існуючий запис у табличці **chambermaid**.

Виконаємо редагування даних в таблиці **room/chambermaid**, змінимо значення chambermaid\_id, яке не існує у табличці **chambermaid**. Спробуємо відредагувати в таблиці **room/chambermaid** рядок з room\_id = 582 та chambermaid\_id = 6 на наступні дані room\_id = 582 та chambermaid\_id = 45.

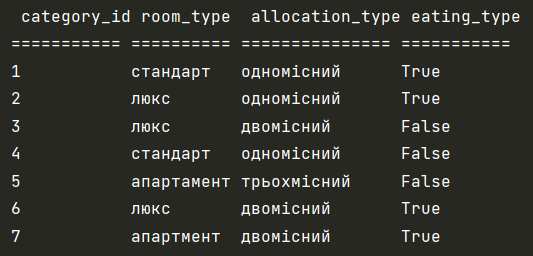
 На екранному відображенні чітко видно, що виникає помилка при редагуванні даних, оскільки зовнішній ключ chambermaid\_id = 45, посилається на неіснуючий запис у табличці **chambermaid**.

**Завдання 2**

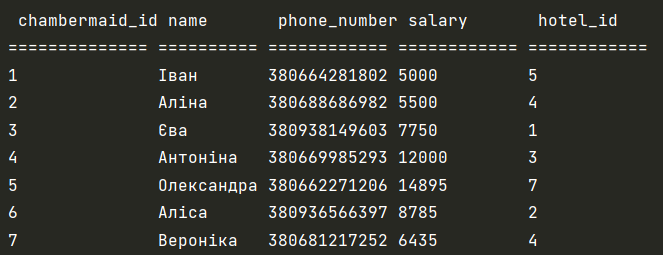
Вставимо по 5 псевдорандомізованих записів у кожну таблицю.

Записи таблиць перед вставкою даних:

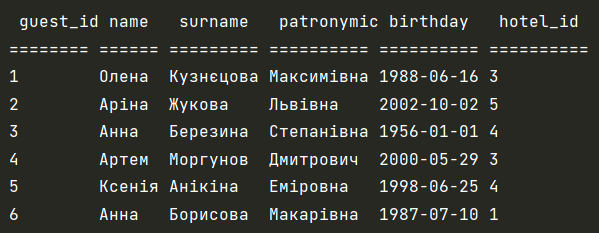
Таблиця **category**:



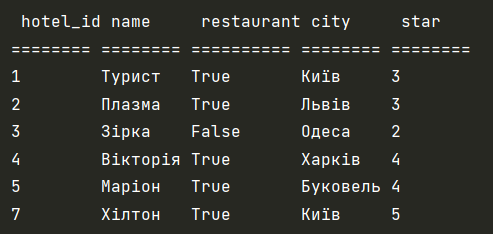
Таблиця **chambermaid**:



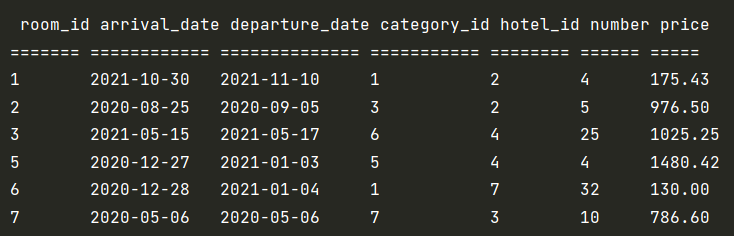
Таблиця **guest**:



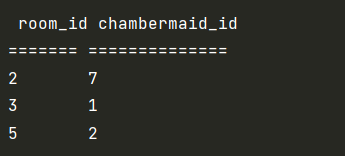
Таблиця **hotel**:



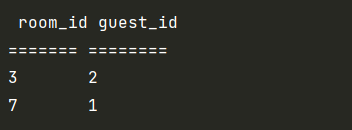
Таблиця **room**:



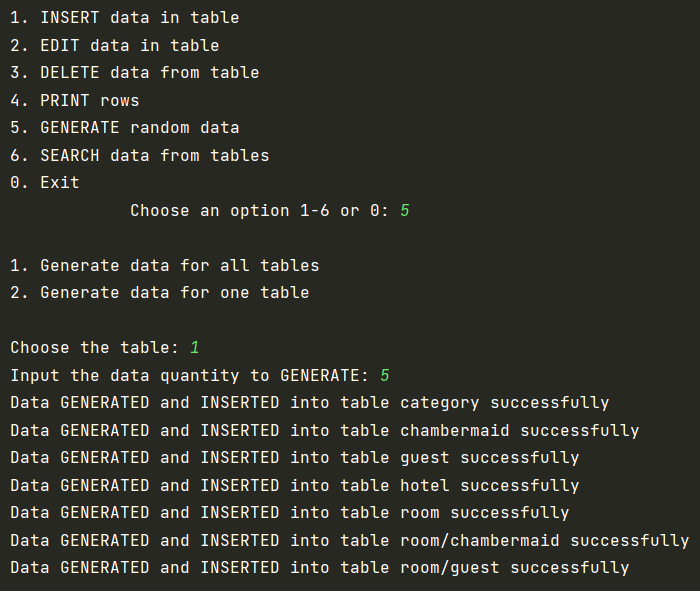
Таблиця **room/chambermaid**:



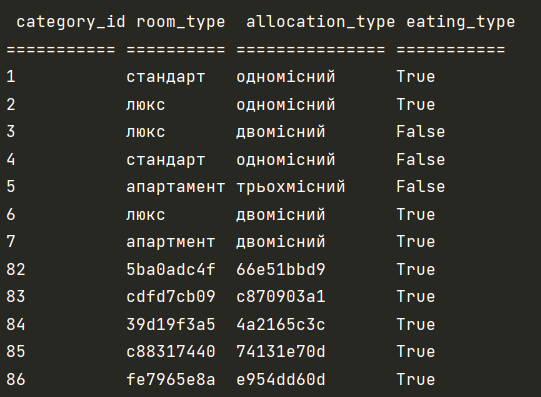
Таблиця **room/guest**:



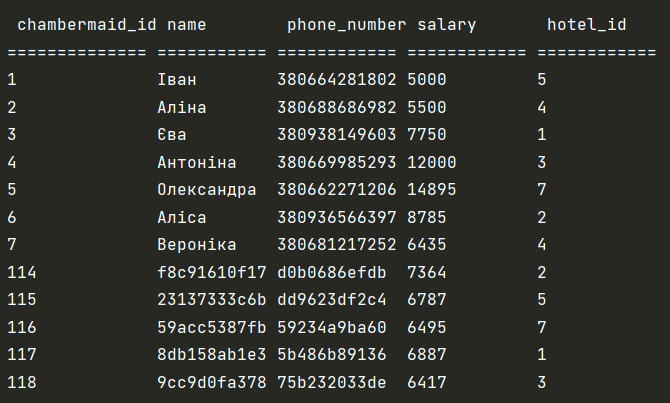
Таблиці після вставки згенерованих записів:



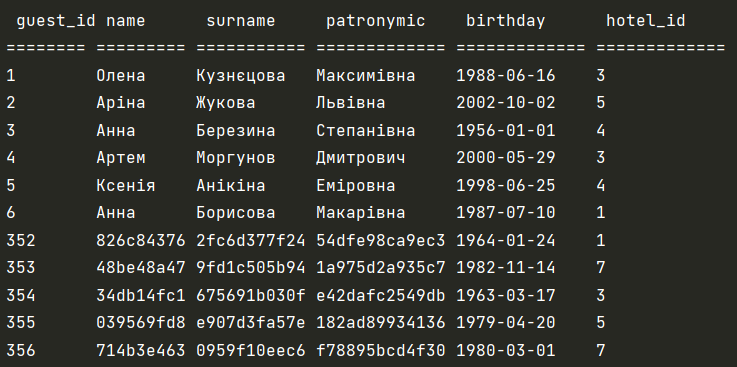
Таблиця **category**:



Таблиця **chambermaid**:



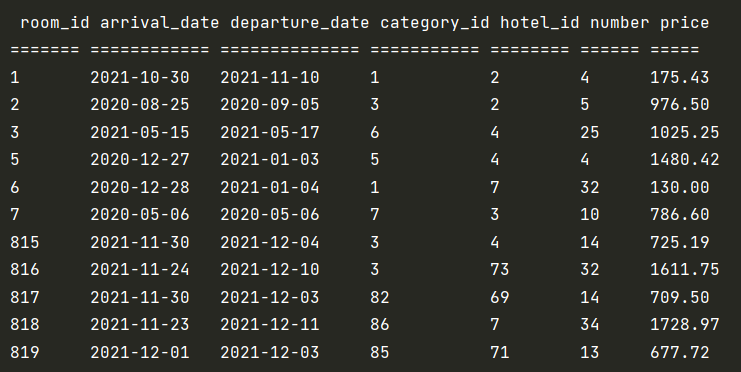
Таблиця **guest**:



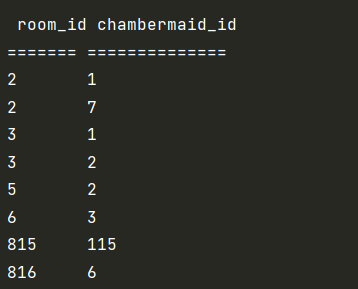
Таблиця **hotel**:



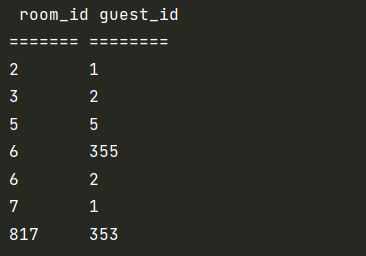
Таблиця **room**:

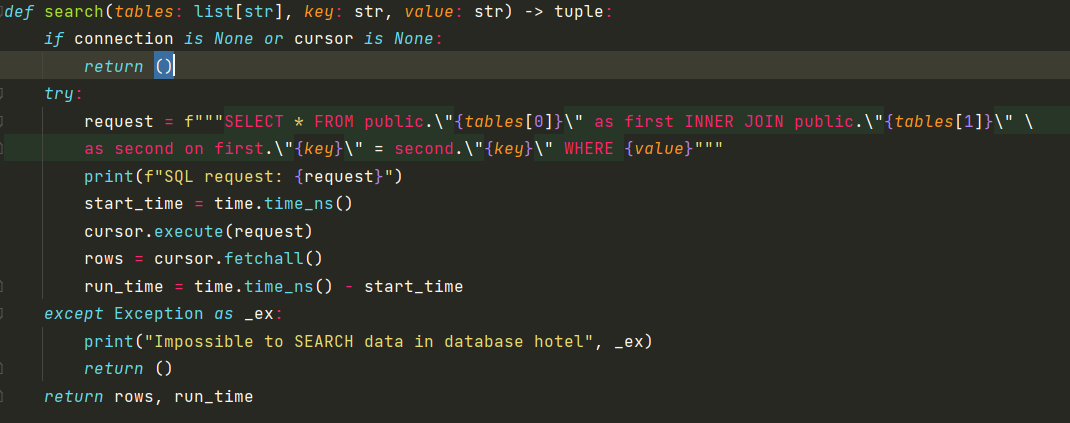


Таблиця **room/chambermaid**:

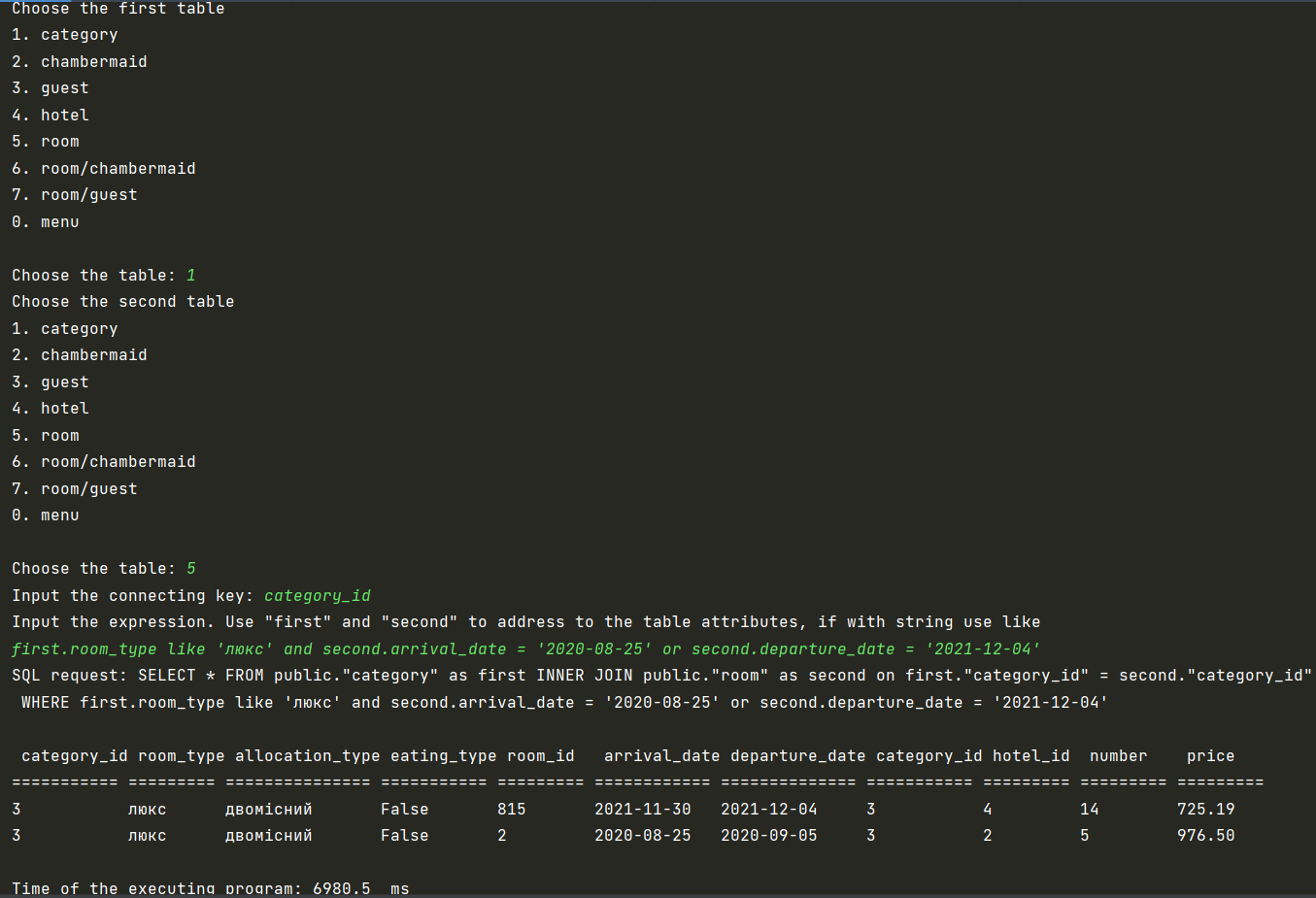


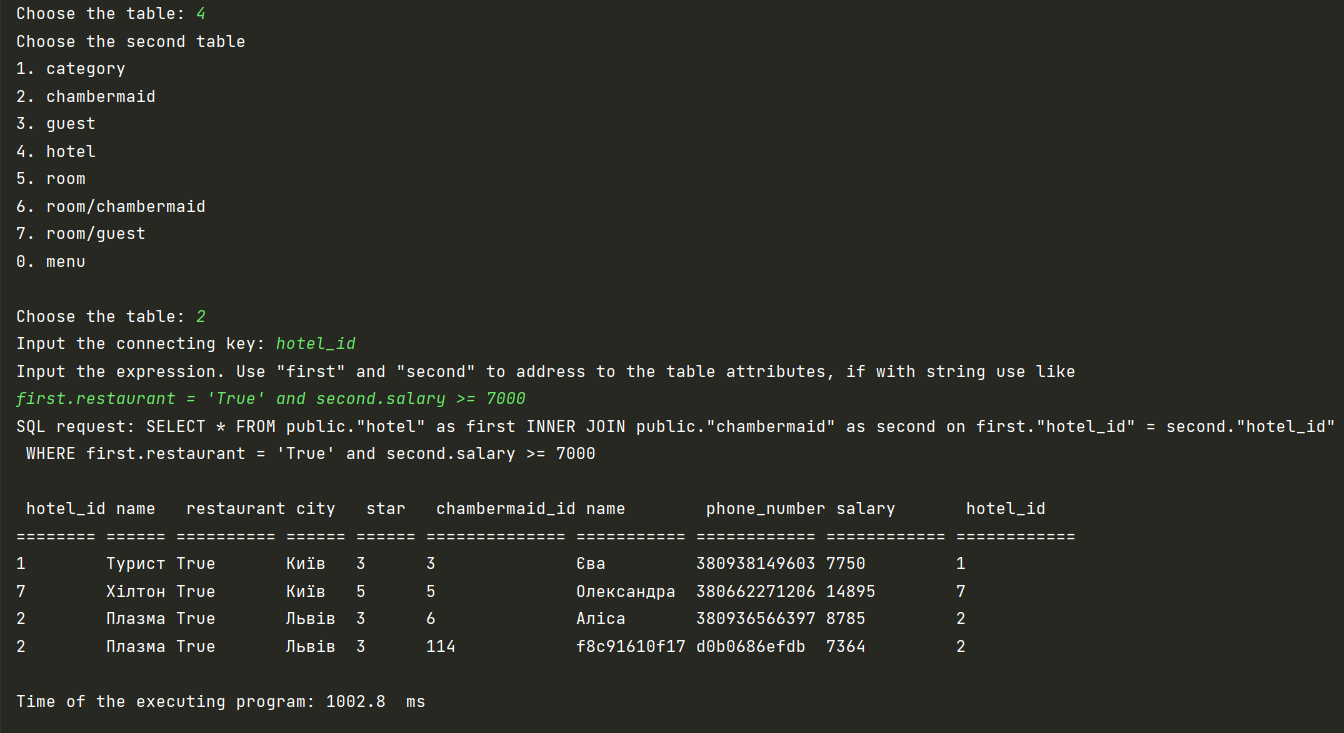
Таблиця **room/guest**:

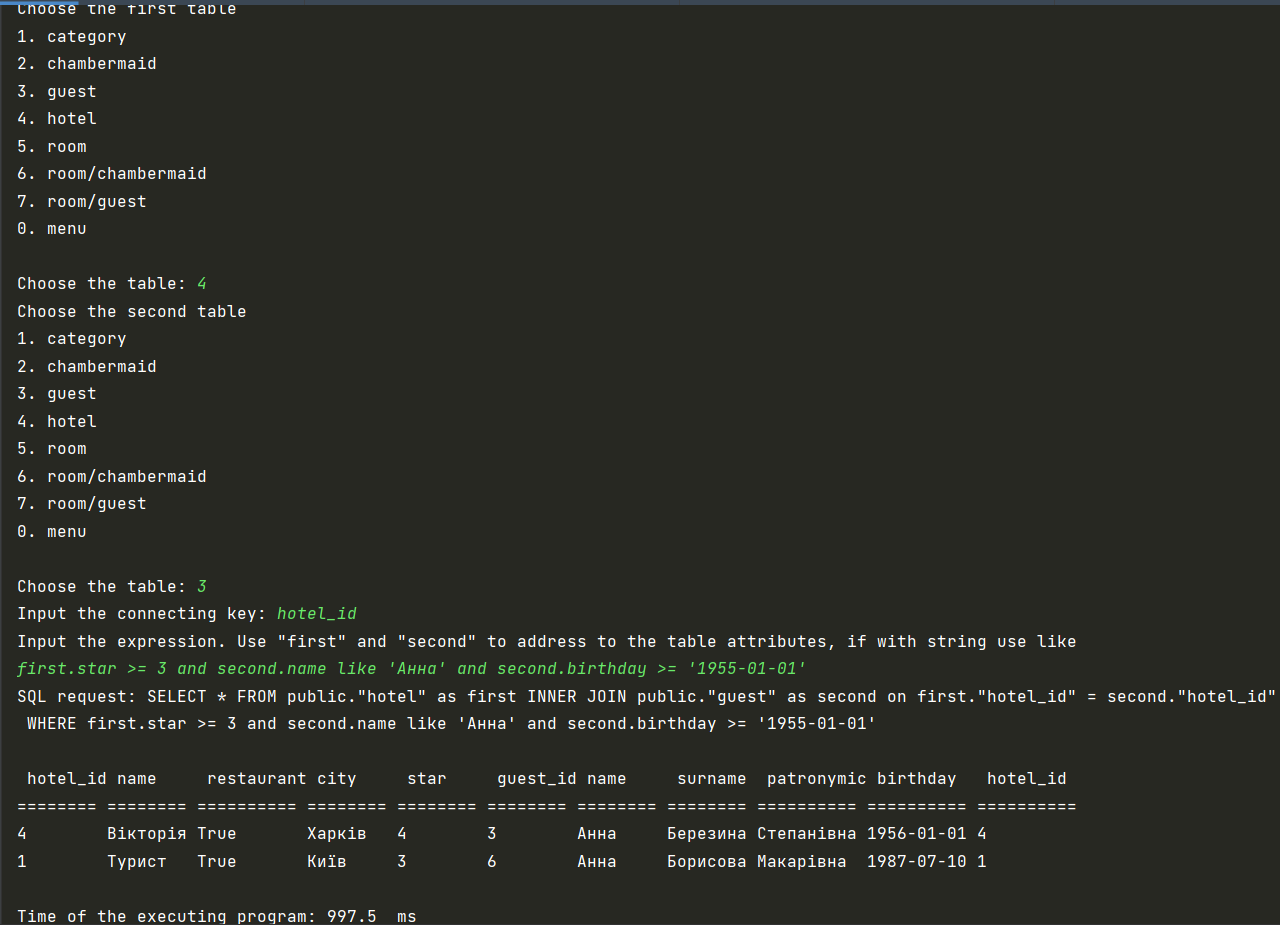


**Завдання 3**

Виконаємо об’єднання двох таблиць та пошук у них за трьома атрибутами:

Пошук перший:

Пошук другий:

Третій пошук:

**Завдання 4**

Код програмного модулю “model.py”

import psycopg2

import time

cursor = None

connection = None

def connect():

try:

global cursor, connection

connection = psycopg2.connect(

host="localhost",

user="postgres",

password="postgres",

database="hotel\_db",

port="5432"

)

cursor = connection.cursor()

print("Successfully CONNECTED to database hotel")

# cursor.execute("SELECT version();")

# print(f"Server version {cursor.fetchone()}")

except Exception as \_ex:

print("Failed CONNECTION to database hotel", \_ex)

def disconnect():

try:

cursor.close()

connection.close()

print("Successfully DISCONNECTED from database hotel")

except Exception as \_ex:

print("Impossible to DISCONNECT from database hotel", \_ex)

def insert(choice: int, data: list) -> bool:

if connection is None or cursor is None:

return False

else:

try:

match choice:

case 1:

cursor.execute(f"""INSERT INTO public.\"category\" (room\_type, allocation\_type, eating\_type) \

VALUES (\'{data[0]}\', \'{data[1]}\', {data[2]});""")

case 2:

cursor.execute(f"""INSERT INTO public.\"chambermaid\" (name, phone\_number, salary, hotel\_id) \

VALUES (\'{data[0]}\', \'{data[1]}\', {data[2]}, {data[3]});""")

case 3:

cursor.execute(f"""INSERT INTO public.\"guest\" (name, surname, patronymic, birthday, hotel\_id) \

VALUES (\'{data[0]}\', \'{data[1]}\', \'{data[2]}\', \'{data[3]}\', {data[4]});""")

case 4:

cursor.execute(f"""INSERT INTO public.\"hotel\" (name, restaurant, city, star) \

VALUES (\'{data[0]}\', {data[1]}, \'{data[2]}\', {data[3]});""")

case 5:

cursor.execute(f"""INSERT INTO public.\"room\"(arrival\_date, departure\_date, category\_id, hotel\_id,\

number,price) VALUES(\'{data[0]}\', \'{data[1]}\', {data[2]}, {data[3]}, {data[4]}, {data[5]});""")

case 6:

cursor.execute(f"""INSERT INTO public.\"room/chambermaid\" (room\_id, chambermaid\_id) \

VALUES ({data[0]}, {data[1]});""")

case 7:

cursor.execute(f"""INSERT INTO public.\"room/guest\" (room\_id, guest\_id) \

VALUES ({data[0]}, {data[1]});""")

connection.commit()

except Exception as \_ex:

print("Impossible to INSERT data into table", \_ex)

return False

return True

def delete(table: str, key\_name: str, key\_val: str) -> bool:

if connection is None or cursor is None:

return False

else:

try:

cursor.execute(f"""DELETE FROM public.\"{table}\" WHERE {key\_name} = \'{key\_val}\';""")

connection.commit()

except Exception as \_ex:

print(f"Impossible to DELETE data from table {table}", \_ex)

return False

return True

def select\_by\_key(table: str, key\_name: str, key\_val: str) -> list:

if connection is None or cursor is None:

return []

else:

try:

cursor.execute(f"""SELECT \* FROM public.\"{table}\" WHERE {key\_name} = \'{key\_val}\';""")

except Exception as \_ex:

print(f"Impossible to SELECT data from table {table} by key {key\_name}", \_ex)

return []

return cursor.fetchall()

def select\_by\_table(table: str, quantity: str = '100', offset: str = '0') -> list:

if connection is None or cursor is None:

return []

else:

try:

if table == 'room/chambermaid' or table == 'room/guest':

cursor.execute(f"""SELECT \* FROM public.\"{table}\" ORDER BY {"room\_id"} \

ASC limit {quantity} offset {offset};""")

else:

cursor.execute(f"""SELECT \* FROM public.\"{table}\" ORDER BY {table + "\_id"} \

ASC limit {quantity} offset {offset};""")

except Exception as \_ex:

print(f"Impossible to SELECT data from table {table}", \_ex)

return []

return cursor.fetchall()

def update(choice: int, data: list, id1: int, id2: int = 0) -> bool:

if connection is None or cursor is None:

return False

else:

try:

match choice:

case 1:

cursor.execute(f"""UPDATE public.\"category\" SET room\_type = \'{data[0]}\', \

allocation\_type = \'{data[1]}\', eating\_type = {data[2]} WHERE category\_id = {id1};""")

case 2:

cursor.execute(f"""UPDATE public.\"chambermaid\" SET name = \'{data[0]}\', phone\_number = \

\'{data[1]}\', salary = {data[2]}, hotel\_id = {data[3]} WHERE chambermaid\_id = {id1};""")

case 3:

cursor.execute(f"""UPDATE public.\"guest\" SET name = \'{data[0]}\', surname = \'{data[1]}\', \

patronymic = \'{data[2]}\',birthday = \'{data[3]}\',hotel\_id = {data[4]} WHERE guest\_id = {id1};""")

case 4:

cursor.execute(f"""UPDATE public.\"hotel\" SET name = \'{data[0]}\', restaurant = {data[1]}, \

city = \'{data[2]}\', star = {data[3]} WHERE hotel\_id = {id1};""")

case 5:

cursor.execute(f"""UPDATE public.\"room\" SET arrival\_date = \'{data[0]}\', departure\_date = \

\'{data[1]}\', category\_id = {data[2]}, hotel\_id = {data[3]}, number = {data[4]}, \

price = {data[5]} WHERE room\_id = {id1};""")

case 6:

cursor.execute(f"""UPDATE public.\"room/chambermaid\" SET room\_id = {data[0]}, \

chambermaid\_id = {data[1]} WHERE room\_id = {id1} AND chambermaid\_id = {id2};""")

case 7:

cursor.execute(f"""UPDATE public.\"room/guest\" SET room\_id = {data[0]}, \

guest\_id = {data[1]} WHERE room\_id = {id1} AND guest\_id = {id2};""")

connection.commit()

except Exception as \_ex:

print("Impossible to UPDATE data into table", \_ex)

return False

return True

def generate(choice: int, count: int) -> bool:

if connection is None or cursor is None:

return False

try:

for i in range(count):

match choice:

case 1:

cursor.execute(f"""INSERT INTO public.\"category\" (room\_type, allocation\_type, eating\_type) \

VALUES (substr(md5(random()::text), 0, 10), substr(md5(random()::text), 0, 10), \

(round(random())::int)::boolean);""")

case 2:

cursor.execute(f"""INSERT INTO public.\"chambermaid\" (name, phone\_number, salary, hotel\_id) \

SELECT (substr(md5(random()::text), 0, 12)), \

(substr(md5(random()::character varying(12)), 0, 12)), \

(floor(random() \* (25000 - 5000 + 1)) + 5000), \

hotel\_id FROM public."hotel" order by random() limit 1;""")

case 3:

cursor.execute(f"""INSERT INTO public.\"guest\" (name, surname, patronymic, birthday, hotel\_id) \

SELECT substr(md5(random()::text), 0, 10), \

substr(md5(random()::character varying(12)), 0, 12), \

substr(md5((random() \* 2)::text), 0, 14), \

to\_timestamp(-286782355 + random() \* 3270071999), \

hotel\_id FROM public."hotel" order by random() limit 1;""")

case 4:

cursor.execute(f"""INSERT INTO public.\"hotel\" (name, restaurant, city, star) \

VALUES (substr(md5(random()::text), 0, 10), (round(random())::int)::boolean, \

substr(md5(random()::text), 0, 8), (floor(random() \* (5 - 1 + 1)) + 1));""")

case 5:

cursor.execute(f"""INSERT INTO public.\"room\"(arrival\_date, departure\_date, number, price, \

category\_id, hotel\_id) \

SELECT NOW() + (random() \* (NOW() - NOW() - '360 days')), \

NOW() + (random() \* (NOW() - NOW() + '360 days')), \

floor(random() \* (1000 - 10 + 1) + 10), random() \* (50000 - 500 ) + 500, \

category\_id, hotel\_id FROM public."category", public."hotel" \

order by random() limit 1;""")

case 6:

cursor.execute(f"""INSERT INTO public.\"room/chambermaid\" (room\_id, chambermaid\_id) \

SELECT room\_id, chambermaid\_id FROM public."room", public."chambermaid" \

order by random() limit 1;""")

case 7:

cursor.execute(f"""INSERT INTO public.\"room/guest\" (room\_id, guest\_id) \

SELECT room\_id, guest\_id FROM public."room", public."guest" \

order by random() limit 1;""")

connection.commit()

except Exception as \_ex:

print("Impossible to GENERATE data to database hotel", \_ex)

return False

return True

def search(tables: list[str], key: str, value: str) -> tuple:

if connection is None or cursor is None:

return ()

try:

request = f"""SELECT \* FROM public.\"{tables[0]}\" as first INNER JOIN public.\"{tables[1]}\" as second on first.\"{key}\" = second.\"{key}\" WHERE {value}"""

print(f"SQL request: {request}")

start\_time = time.time\_ns()

cursor.execute(request)

rows = cursor.fetchall()

run\_time = time.time\_ns() - start\_time

except Exception as \_ex:

print("Impossible to SEARCH data in database hotel", \_ex)

return ()

return rows, run\_time

Даний модуль є точкою доступу до бази даних з програми. Саме в ньому реалізуються всі запити. Для цього в ньому використовується бібліотека – psycopg2.

Функція connect() – намагається підключитись до БД.

Функція disconnect() – намагається відключитись від БД.

Функція insert(choice, data) – намагається вставити в таблицю по номеру choice дані зі списку data.

Функція delete(table, key\_name, key\_val) – намагається видали з таблиці table рядок з ключем key\_name, який == key\_val.

Функція select\_by\_key(table, key\_name, key\_val) – намагається взяти дані з таблиці table рядок з ключем key\_name, який == key\_val.

Функція select\_by\_table(table, quantity, offset) – намагається взяти дані з таблиці table та відсортувати їх у порядку зростання по primary key.

Функція update(choice, data, id1, id2) – намагається змінити в таблиці choice, даними зі списку data, рядок з первинним ключем, який == id, якщо первинний ключ складений, тоді має дорівнювати id1 та id2.

Функція generate(choice, count) – генерує дані і намагається вставити в таблицю choice, count разів ці даніазрр.

Функція search(tables, key, value) ­– пробує об’єднати і взяти дані з таблиць.