

**Національний технічний університет України**

**“Київський політехнічний інститут”**

**Факультет прикладної математики**

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

**Лабораторна робота №2**

***з дисципліни “* *Бази даних і засоби управління”***

*Тема: “Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”*

Виконав:

студент ІII курсу

групи КВ-01

Таранич Артем

Перевірив: Павловський В. І.

Київ – 2022

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. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі. 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).

Деталізоване завдання:

Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати вилучення рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні внесення нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.

Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими не мовою програмування, а відповідним SQL-запитом!

Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.

Програмний код організувати згідно шаблону Model-View-Controller(MVC). Приклад організації коду згідно шаблону доступний за даним посиланням. При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати лише мову SQL.

**Логічна модель предметної області «Аеропорт»**

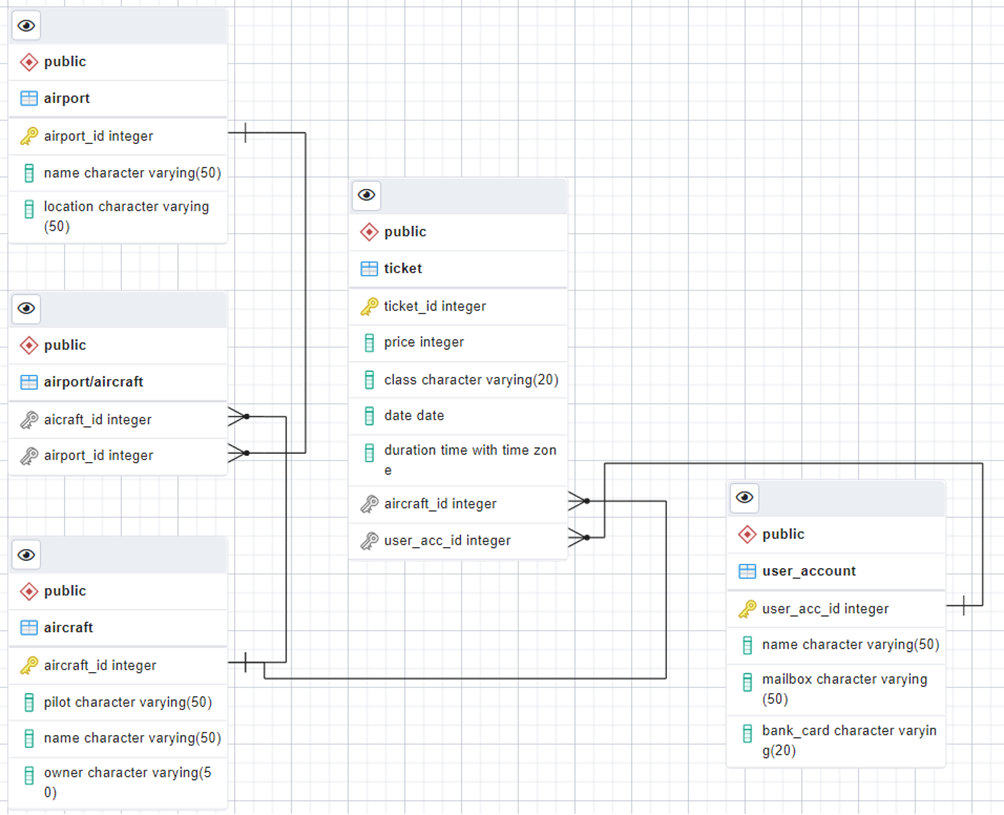


Рисунок 1. Схема бази даних, побудовано у [p](https://app.diagrams.net/)gAdmin 4.

**Середовище та компоненти розробки**

Для розробки використовувалась мова програмування Python, середовище розробки Visual Studio Code, а також стороння бібліотека, що надає API для доступу до PostgreSQL – psycopg2.

**Шаблон проектування**

MVC - Шаблон проектування, який використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Згідно компоненту моделі, у моїй програмі відповідають всі компоненти які знаходять у файлі model.py.

View – в нашому випадку консольний інтерфейс з яким буде взаємодіяти наш користувач. Згідно компоненту представлення, то їй відповідають такі компоненти, згідно яким користувач бачить необхідні дані, що є представленням даних у вигляді консольного інтерфейсу.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує введені користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок.

**Структура програми та її опис**

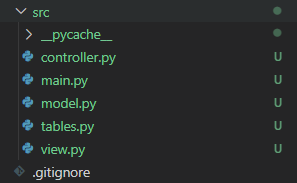


Рисунок 2. Структура програми.

Програма умовно поділена на 4 модулі: main.py, model.py, view.py, controller.py.

Код у файлі controller.py описує клас Controller, який взаємодія з користувачем, передає команди модулю Model та відправляє отримані дані в View.

Код у файлі model.py описає клас Model, що займається регулювання підключення до бази даних, та виконанням низькорівневих запитів до неї.

Код у файлі view.py описує клас View, що виводить результати виконання.

Файл main.py це початкова точка програми, яка створює екземпляр класу Controller і надалі користувач взаємодіє вже з контроллером.

**Меню операцій складається з восьми пунктів**

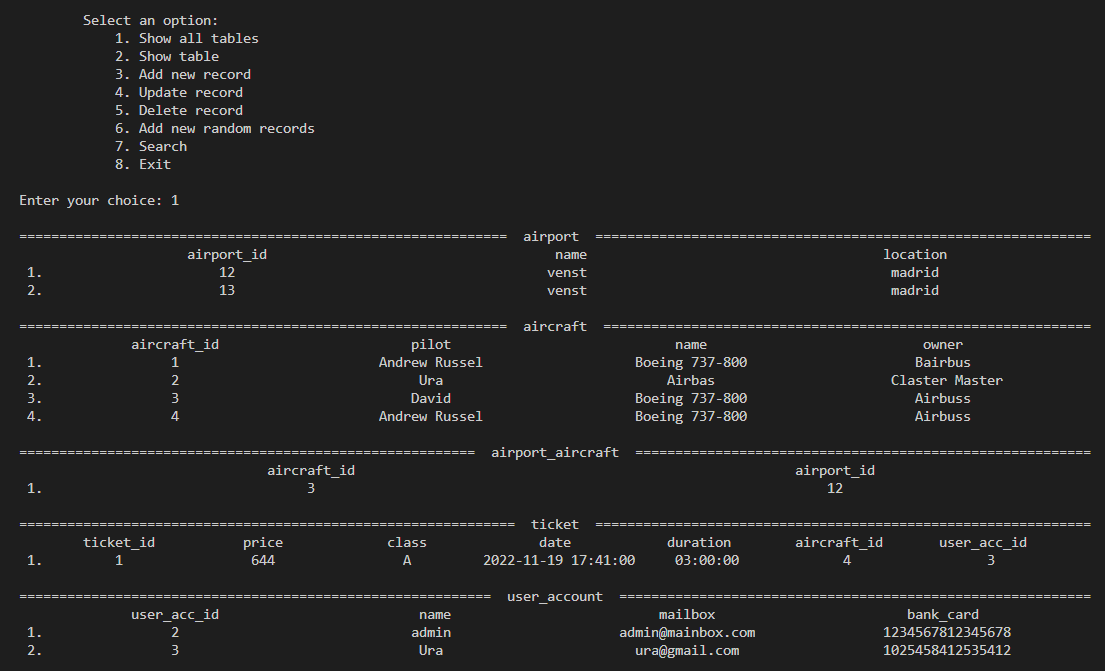
1. Виведення усіх таблиць в консоль
2. Виведення вибраної таблиці в консоль.
3. Додавання нового рядку до таблиці.
4. Оновлення рядку у таблиці.
5. Видалення рядку з таблиці.
6. Додавання нових випадкових даних до кожної таблиці БД.
7. Пошук усіх рядків з вибраної таблиці, по параметрам, які ввів користувач.
8. Завершення програми.

**Меню таблиць складається з 5 пунктів**

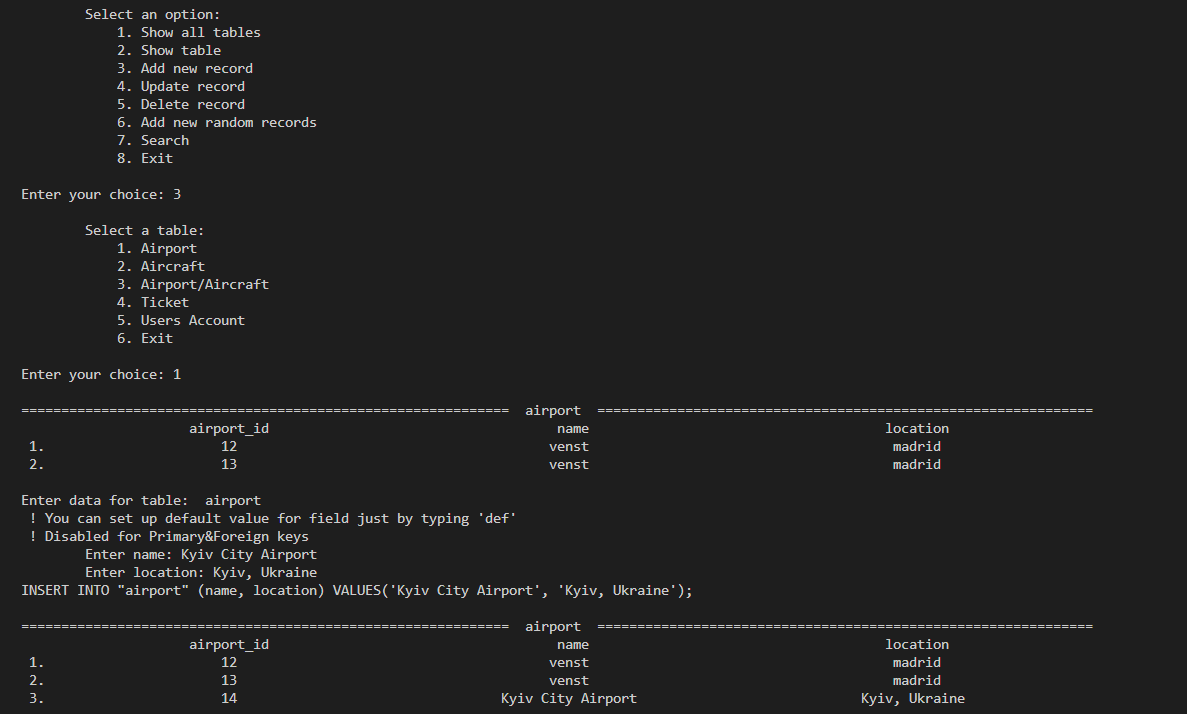
1. Виконати обрану операцію над таблицею “Airport”.
2. Виконати обрану операцію над таблицею “Aircraft”.
3. Виконати обрану операцію над таблицею “Airport/Aircraft”.
4. Виконати обрану операцію над таблицею “Ticket”.
5. Виконати обрану операцію над таблицею “Users Account”.

**Результати та виконання операцій**

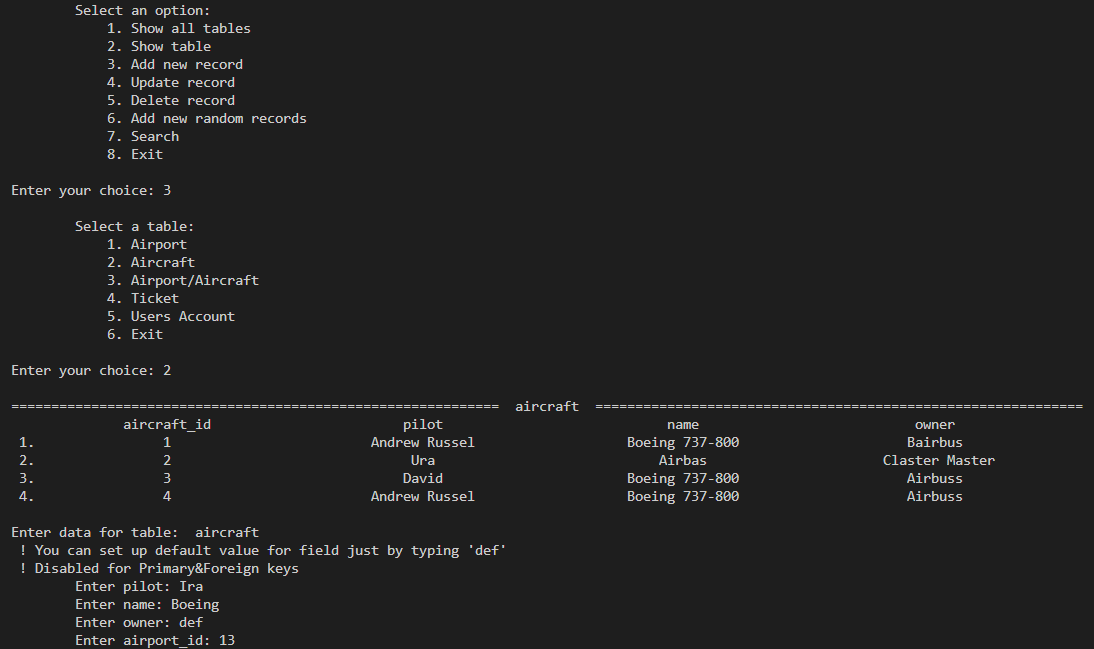
**Виведення усіх таблиць в консоль**

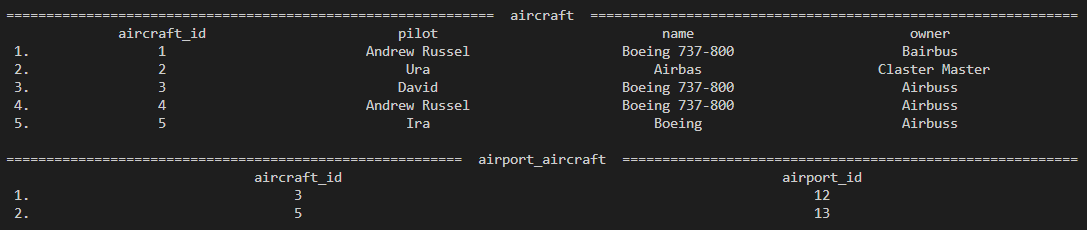
****

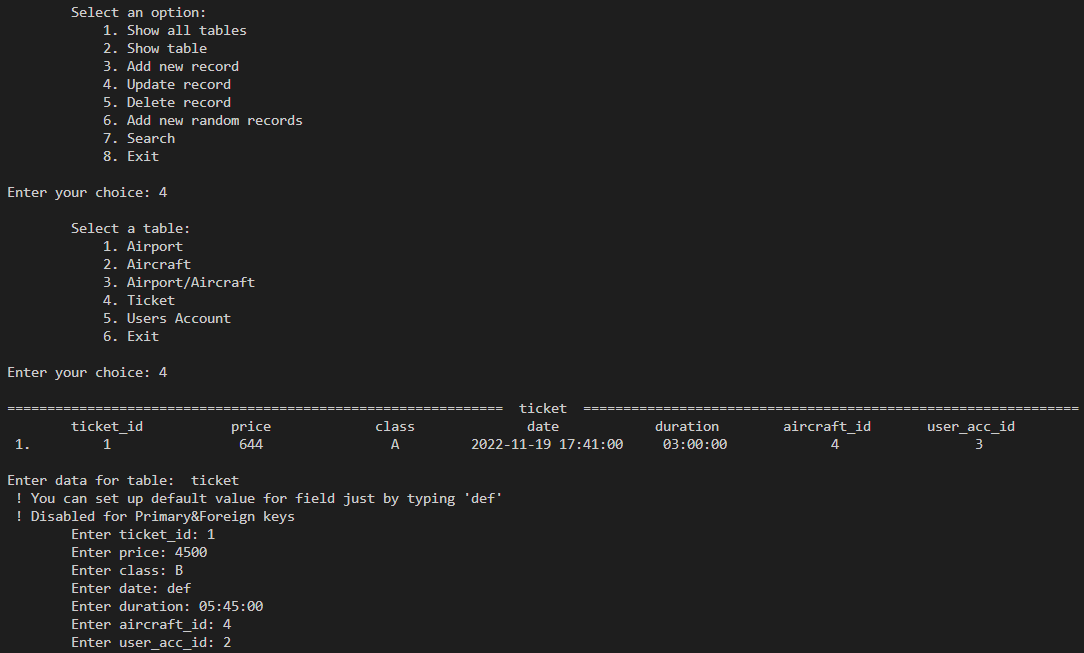
**Додавання нового рядку до таблиці “Aiport”**

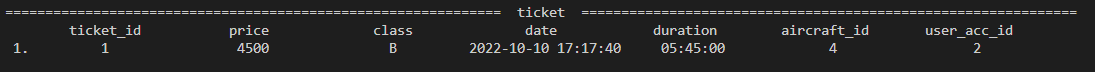
****

**Додавання нового рядку до таблиці “Aircraft”**

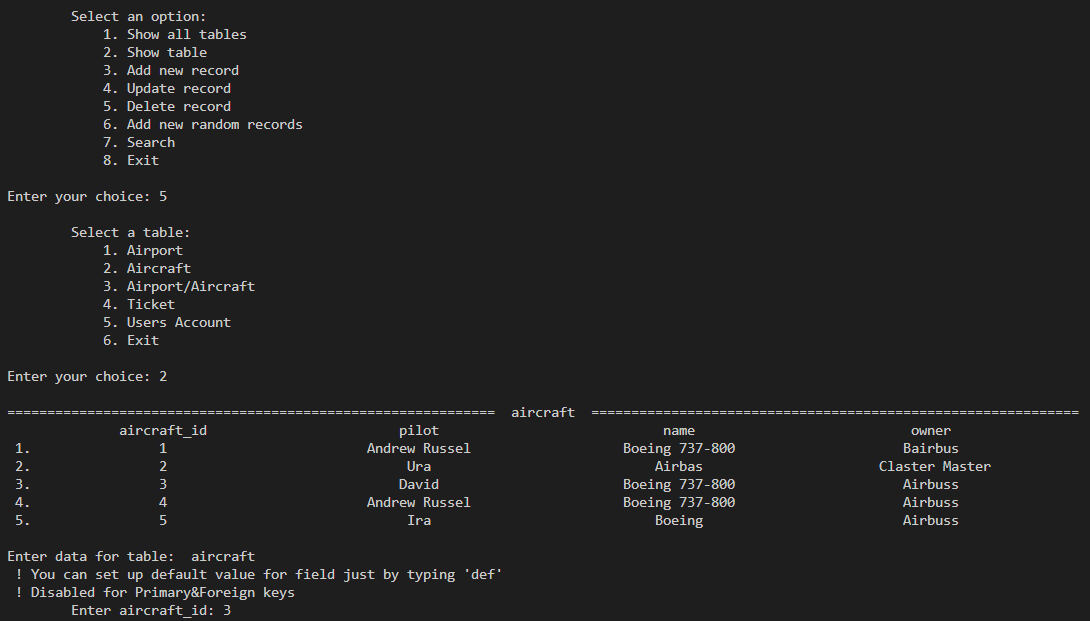
В результаті дані також оновилися в зв’язаній таблиці “Airport/Aircraft”

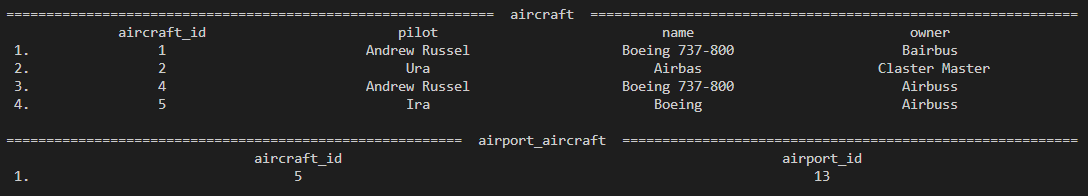
****

**Оновлення рядку у таблиці “Ticket”**

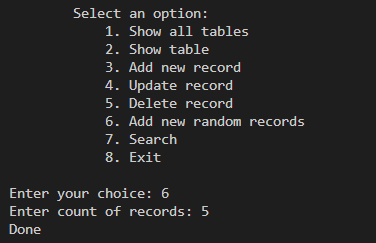


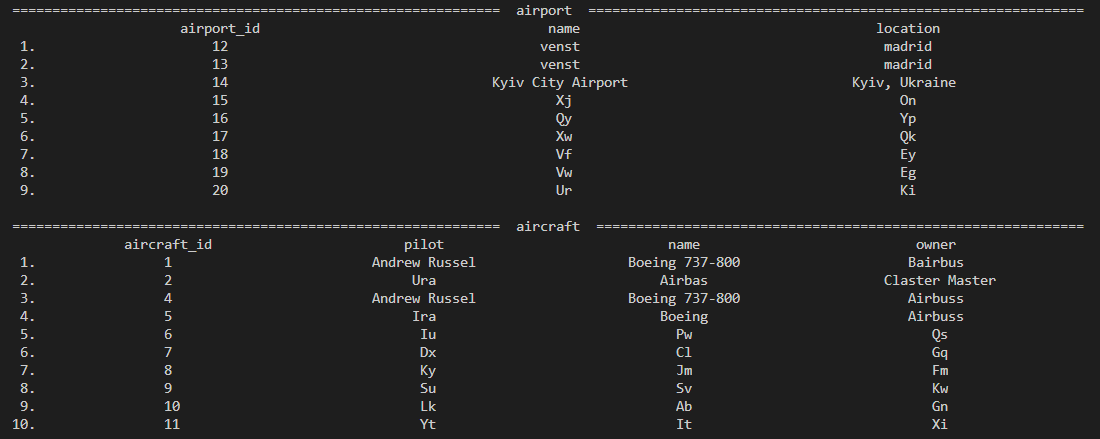
**Видалення рядку з таблиці “Aircraft”**

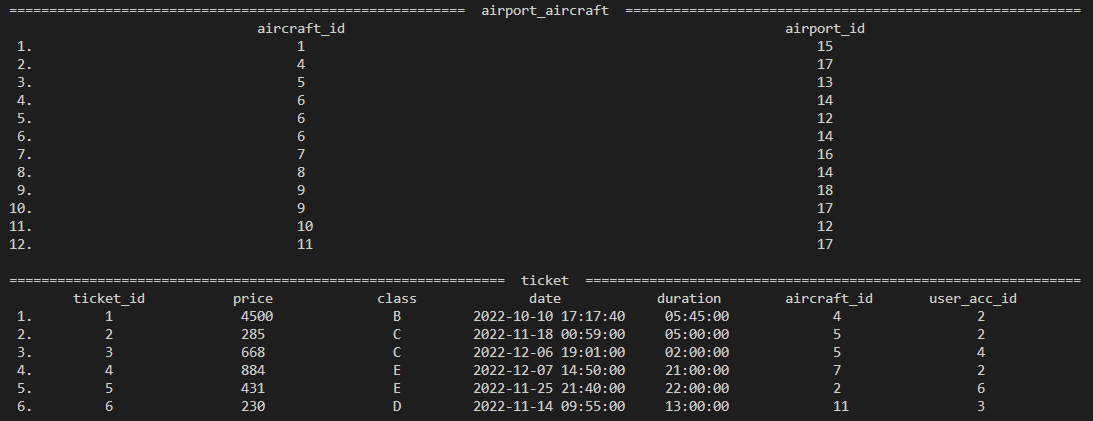
В результаті дані також оновилися в зв’язаній таблиці “Airport/Aircraft”

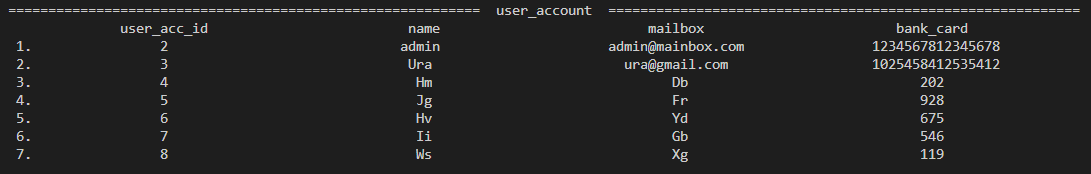
****

**Додавання нових випадкових даних до кожної таблиці БД.**

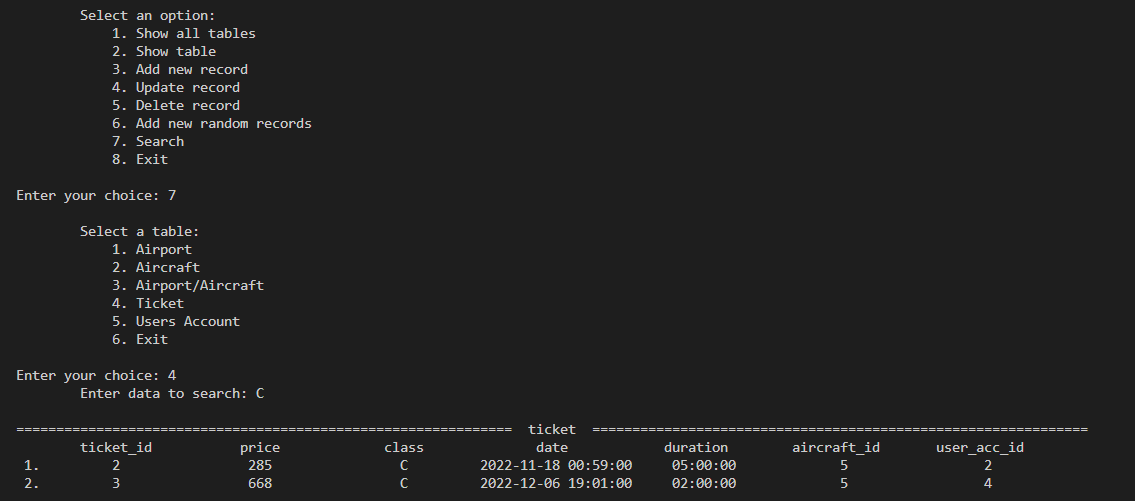








**Пошук усіх рядків з вибраної таблиці, по параметрам, які ввів користувач.**

****

**Код програми**

**main.py**

from controller import Controller

obj = Controller()

obj.menu()

**model.py**

from random import choice

import psycopg2

import psycopg2.extras

from tables import tablesList, table

class Model:

def \_\_init\_\_(self):

self.connection = psycopg2.connect(

host ='localhost',

user = "postgres",

password = 'qwerty',

database = 'airport',

)

self.connection.autocommit = True

def select(self, table: table):

with self.connection.cursor() as cursor:

cursor.execute(f"SELECT \* FROM \"{table.name}\" " +

f"ORDER BY \"{table.id}\" ASC")

return cursor.fetchall()

def insert\_into\_table(self, table: table, data: list):

match table.name:

case 'aircraft':

airport\_id = data.pop()

self.insert\_into(table, data) # insert into aircraft

aircraft\_id = self.select(table)[-1][0]

self.insert\_into(tablesList[2], [aircraft\_id, airport\_id]) # insert into airport\_aircraft

case \_:

self.insert\_into(table, data)

def insert\_into(self, table: table, data):

parsed\_colms = ''

for inserted\_column in table.changing\_columns:

parsed\_colms += f'{inserted\_column}' + ', '

parsed\_colms = parsed\_colms[:-2]

inserted\_data = ''

for value in data:

inserted\_data += f'\'{value}\'' + ', '

inserted\_data = inserted\_data[:-2]

with self.connection.cursor() as cursor:

cursor.execute(f"INSERT INTO \"{table.name}\" ({parsed\_colms}) " +

f"VALUES({inserted\_data});")

def update\_table(self, table: table, data):

setter = "SET "

columns = table.changing\_columns

id = data.pop(0)

if len(columns) != len(data): raise Exception('Wrong data input')

for i in range(0, len(columns)):

setter += f'"{columns[i]}" = \'{data[i]}\', '

setter = setter[:-2]

with self.connection.cursor() as cursor:

cursor.execute(f"UPDATE \"{table.name}\" " +

f"{setter} " +

f"WHERE \"{table.id}\" = {id};")

def delete\_table(self, table: table, data):

with self.connection.cursor() as cursor:

cursor.execute(f"DELETE FROM \"{table.name}\" " +

f"WHERE {table.id} = {data[0]};")

def search\_in\_table(self, table: table, data):

searcher = "WHERE "

for column in table.columns:

searcher += f'\"{column}\"::text LIKE \'%{data}%\' OR '

searcher = searcher[:-4]

with self.connection.cursor() as cursor:

cursor.execute(f"SELECT \* FROM \"{table.name}\" " +

f"{searcher};")

return cursor.fetchall()

def insert\_random\_data(self, count: int):

for i in range(0, count):

with self.connection.cursor() as cursor:

inserted\_data = [

self.generate\_random\_str(),

self.generate\_random\_str(),

]

self.insert\_into\_table(tablesList[0], inserted\_data) # insert into airport

inserted\_data = [

self.generate\_random\_str(),

self.generate\_random\_str(),

self.generate\_random\_str(),

choice(self.get\_ids\_from\_table(tablesList[0]))[0],

]

self.insert\_into\_table(tablesList[1], inserted\_data) # insert into aircraft

inserted\_data = [

choice(self.get\_ids\_from\_table(tablesList[1]))[0],

choice(self.get\_ids\_from\_table(tablesList[0]))[0],

]

self.insert\_into\_table(tablesList[2], inserted\_data) # insert into airport\_aircraft

inserted\_data = [

self.generate\_random\_int(),

choice(['A', 'B', 'C', 'D', 'E']),

self.generate\_random\_time(),

self.generate\_random\_duration(),

choice(self.get\_ids\_from\_table(tablesList[1]))[0],

choice(self.get\_ids\_from\_table(tablesList[4]))[0],

]

self.insert\_into\_table(tablesList[3], inserted\_data) # insert into tiket

inserted\_data = [

self.generate\_random\_str(),

self.generate\_random\_str(),

self.generate\_random\_int(),

]

self.insert\_into\_table(tablesList[4], inserted\_data) # insert into user\_account

def generate\_random\_str(self):

uppercase\_letter = "chr(ascii('A') + (random() \* 25)::int) "

lowercase\_letter = "chr(ascii('a') + (random() \* 25)::int) "

with self.connection.cursor() as cursor:

cursor.execute(f"SELECT ({uppercase\_letter} || {lowercase\_letter});")

return cursor.fetchone()[0]

def generate\_random\_int(self):

with self.connection.cursor() as cursor:

cursor.execute(f"SELECT trunc(random() \* 1000)::int")

return cursor.fetchone()[0]

def generate\_random\_time(self):

with self.connection.cursor() as cursor:

cursor.execute("""SELECT timestamp '2022-11-10 13:00:00'

+ make\_interval(days => (random() \* 30)::int)

+ make\_interval(hours => (random() \* 45)::int)

+ make\_interval(mins => (random() \* 59)::int)

""")

return cursor.fetchone()[0]

def generate\_random\_duration(self):

with self.connection.cursor() as cursor:

cursor.execute("SELECT time '1:00:00' + make\_interval(hours => (random() \* 24)::int)")

return cursor.fetchone()[0]

def get\_ids\_from\_table(self, table: table):

with self.connection.cursor() as cursor:

cursor.execute(f"SELECT \"{table.id}\" FROM \"{table.name}\";")

return cursor.fetchall()

**view.py**

from tables import table

class View:

def \_\_init\_\_(self):

self.table = None

def menu(self):

print('''

Select an option:

1. Show all tables

2. Show table

3. Add new record

4. Update record

5. Delete record

6. Add new random records

7. Search

8. Exit

''')

def list\_of\_tables(self):

print('''

Select a table:

1. Airport

2. Aircraft

3. Airport/Aircraft

4. Ticket

5. Users Account

6. Exit

''')

def show\_table(self, table: table, data):

columns = table.columns

column\_width = int(130 / len(columns))

executor\_width = 4

print("\n" + f' {table.name} '.center(130 + executor\_width, "="), end='\n ')

for column in columns:

print(str(column).center(column\_width," "), end='')

print()

for i, item in enumerate(data, start = 1):

print(str(f'{i}.').center(executor\_width," "), end='')

for j in item:

print(str(j).center(column\_width, " "), end='')

print()

**controller.py**

from model import Model

from view import View

from datetime import datetime

from tables import tablesList, table

class Controller:

def \_\_init\_\_(self):

self.view = View()

self.model = Model()

def menu(self):

while True:

try:

self.view.menu()

choice = int(input('Enter your choice: '))

match choice:

case 1: # show all tablesList

for i in range(0, len(tablesList)):

data = self.model.select(tablesList[i])

self.view.show\_table(tablesList[i], data)

case 2: # show table

table = self.choose\_table()

data = self.model.select(table)

self.view.show\_table(table, data)

case 3: # add new record

table = self.choose\_table()

self.view.show\_table(table, self.model.select(table))

self.model.insert\_into\_table(table, self.input\_data(table, 'insert'))

self.view.show\_table(table, self.model.select(table))

case 4: # update record

table = self.choose\_table()

self.view.show\_table(table, self.model.select(table))

self.model.update\_table(table, self.input\_data(table, 'update'))

self.view.show\_table(table, self.model.select(table))

case 5: # delete record

table = self.choose\_table()

self.view.show\_table(table, self.model.select(table))

self.model.delete\_table(table, self.input\_data(table, 'delete'))

self.view.show\_table(table, self.model.select(table))

case 6: # add new random records

count = int(input('Enter count of records: '))

self.model.insert\_random\_data(count)

print('Done')

case 7: # search

table = self.choose\_table()

result = self.model.search\_in\_table(table, input('\tEnter data to search: '))

self.view.show\_table(table, result)

case 8: # exit

exit()

case \_: print('Invalid choice, retry')

except ValueError as err: print('Exception: ', err)

def choose\_table(self):

self.view.list\_of\_tables()

while True:

try:

choice = int(input('Enter your choice: '))

match choice:

case 1: # Airport

return (tablesList[0])

case 2: # Aircraft

return (tablesList[1])

case 3: # airport\_aircraft

return (tablesList[2])

case 4: # Ticket

return (tablesList[3])

case 5: # User Account

return (tablesList[4])

case 6: # exit

self.menu()

exit()

case \_: print('Invalid choice, retry')

except ValueError as err: print('Exception: ', err)

def input\_data(self, table: table, instruction):

data = []

print('\nEnter data for table: ', table.name)

print(" ! You can set up default value for field just by typing 'def'")

print(" ! Disabled for Primary&Foreign keys")

columns = []

index = 0

match instruction:

case 'insert':

columns = table.get\_all\_columns()

case 'update':

index = 1

columns = table.columns

case 'delete':

index = 1

columns = [table.columns[0]]

if instruction == 'update' and table.name == 'airport\_aircraft':

data.append(input('\tEnter current aircraft\_id: '))

for column in columns:

data.append(input('\tEnter ' + column + ': '))

if instruction == 'delete': return data

match table.name:

case 'airport': # Airport

if data[index] == 'def': data[index] = 'London City Airport'

index += 1

if data[index] == 'def': data[index] = 'Newham, London, UK'

case 'aircraft': # Aircraft

if data[index] == 'def': data[index] = 'Andrew Russel'

index += 1

if data[index] == 'def': data[index] = 'Boeing 737-800'

index += 1

if data[index] == 'def': data[index] = 'Airbuss'

index += 1

if instruction == 'insert' and data[index] == 'def':

print('Please retry input data, for this coloumn default option is disabled')

return self.input\_data(table)

case 'airport\_aircraft': # airport\_aircraft

if data[0] == 'def' or data[1] == 'def':

print('Please retry input data, for this table default option is disabled')

return self.input\_data(table)

case 'ticket': # Ticket

if data[index] == 'def': data[index] = '1000'

index += 1

if data[index] == 'def': data[index] = 'E'

index += 1

if data[index] == 'def': data[index] = '2022-10-10 ' + datetime.now().strftime("%H:%M:%S")

index += 1

if data[index] == 'def': data[index] = '05:00:00'

index += 1

if data[index] == 'def' or data[index+1] == 'def':

print('Please retry input data, for foreign keys default option is disabled')

return self.input\_data(table)

case 'user\_account': # User Account

if data[index] == 'def': data[index] = 'admin'

index += 1

if data[index] == 'def': data[index] = 'admin@mainbox.com'

index += 1

if data[index] == 'def': data[index] = '0000000000000000'

case \_:

print('Invalid choice, retry')

return self.input\_data(table)

return data

def input\_data\_to\_search(self):

return input('\tEnter data to search: ')

**tables.py**

airport\_table = [

'airport\_id',

'name',

'location'

]

aircraft\_table = [

'aircraft\_id',

'pilot',

'name',

'owner',

]

airport\_aircraft\_table = [

'aircraft\_id',

'airport\_id',

]

ticket\_table = [

'ticket\_id',

'price',

'class',

'date',

'duration',

'aircraft\_id',

'user\_acc\_id',

]

user\_account\_table = [

'user\_acc\_id',

'name',

'mailbox',

'bank\_card',

]

class table:

def \_\_init\_\_(self, name: str, columns: list, associated\_keys: list = None,):

self.name = name

self.columns = columns.copy()

self.id = columns[0]

if name != 'airport\_aircraft':

columns.pop(0)

self.changing\_columns = columns

self.associated\_keys = None

if associated\_keys is not None:

self.associated\_keys = associated\_keys.copy()

def get\_table\_id(self):

return self.columns[0]

def get\_all\_columns(self):

columns = self.changing\_columns.copy()

if self.associated\_keys is not None:

columns += self.associated\_keys

return columns

tablesList = [

table('airport', airport\_table),

table('aircraft', aircraft\_table, [airport\_aircraft\_table[1]]),

table('airport\_aircraft', airport\_aircraft\_table),

table('ticket', ticket\_table),

table('user\_account', user\_account\_table),

]