

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

**Лабораторна робота № 2**

з дисципліни «Бази даних і засоби управління»

«**Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL**»

Виконав: Мазенко Михайло

Студент групи КВ-93

Перевірив: Павловський В.І.

Київ 2021

**Лабораторна робота №2**

Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL

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

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції 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).

**Ссс**

**Середовище розробки**

Середовище розробки програми – Visual Studio. Мова програмування Python3.6.

Середовище розробки бази даних - PostgreSQL

Використані бібліотеки:

* psycopg2 (для зв’язку з СУБД);
* datetime (для роботи з датою і передачею її у запити до БД);
* time (для виміру часу запиту пошуку для завдання 3);
* sys (для реалізації консольного інтерфейсу).

Посилання на репозиторій у GitHub з вихідним кодом програми та звітом: https://github.com/Qlevit8/BDIZU\_lab

**Логічна модель «Бібліотека»**

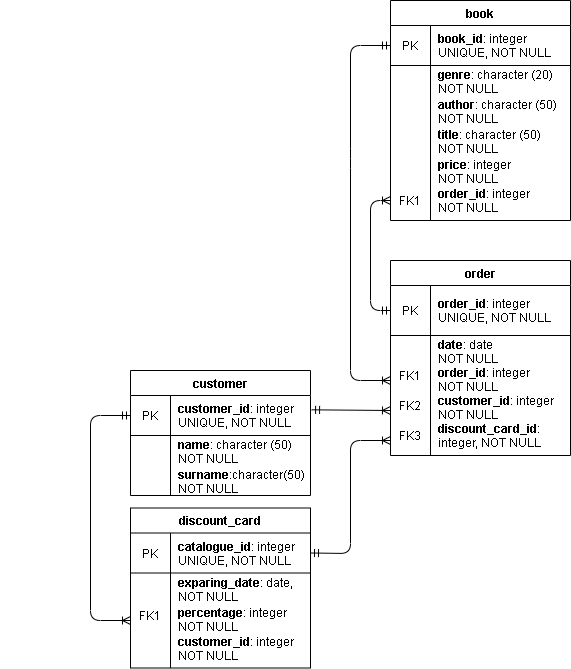


Рисунок 1 – Логічна модель БД

**Шаблон проектування**

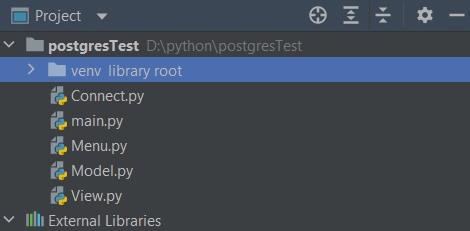
MVC - Шаблон проектування, який використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Даний модуль є точкою доступу до бази даних з програми. Саме в ньому реалізуються всі запити. Це файл model.py. Для цього в ньому використовується бібліотека мови Python – psycopg2. Конструктор класу Model налагоджує зв’язок із сервером і видає повідомлення про помилку, якщо зв’язок не було встановлено.

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

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, наприклад, у вигляді подання. Згідно компоненту контролер, то йому відповідає файл Controller.py.

**Структура програми та її опис**



Файл Model.py

Метод GetTables() не приймає аргументів, повертає список з назвами всіх таблиць в базі даних.

Метод GetFields(string) приймає назву таблиці в якоісті аргументу, повертає список з назвами всіх стопців цієї таблиці.

Метод IfInDatabase(string) перевіряє чи є таблиця з заданим іменем в базі даних.

Метод ShowTable(string) приймає назву таблиці, та викликає функцію виводу цієї таблиці в консоль.

Метод ShowAllTables() не приймає аргументів, та викликає функцію ShowTable для кожної таблиці.

Метод InsertInTable(string) приймає назву таблиці, користувач вибирає параметри для внесення в цю таблицю.

Метод DeleteFromTable(string) приймає назву таблиці, знищує рядок за наданою користувачем ознакою.

Метод UpdateTable(string) приймає назву таблиці, оновлює вибрані користувачем записи заданого рядка.

Метод IsIdentity(string, string, cursor) приймає назву таблиці, колонку та курсор, перевіряє чи задана колонка підтримує лише унікальні записи.

Метод GetСolumnType(string, string) приймає назву таблиці та колонки, повертає тип колонки.

Метод GenerateDataForTable(string, int) приймає назву таблиці та кількість рядків, які потрібно згенерувати та вставити в таблицю.

Метод FindSameColumns(string, string) приймає дві таблиці та знаходить спільні за назвою колонки, поміщує їх в список та повертає.

Метод SearchForDataInTables() не приймає аргументів, користувач може вибрати за якими ознаками буде виконуватись пошук. Пошук може бути виконаний лише за двома або більшою кількістю задіяних таблиць.

Файл View.py

Метод GetInstruction() виводить інструкцію щодо користування інтерфейсом програми.

Метод CheckIfFk(string, string) приймає назву таблиці та колонки, перевіряє чи є ця колонка зовнішнім ключем.

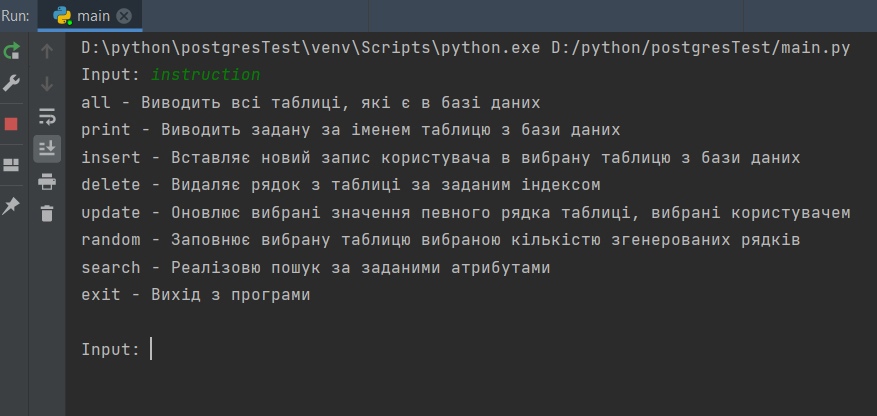
Метод ChooseOperationByType(string, string/list , string, string) приймає назви типу, значення, назви, яка присвоєнна цьому елементу та назву колонки, повертає рядок для подальшого використання в sql запиті.

Метод FindType(string) приймає назву типу, повертає рядок для подальшого використання в sql запиті.

Файл Menu.py

Метод MainMenu() є головним меню, через яке користувач може взаємодіяти з інтерфейсом програми за допомогою ключових слів.

**Структура меню програми**

 Рисунок 2 – Результат команди instruction

На знімку екрану терміналу продемонстовано виконання команди instruction, що показує усі доступні користувачу команди, коротко описує їх та надає список обов’язкових аргументів. Кожна команда запускає відповідний метод об’єкту класу Controller, який реалізує передачу аргументів у клас View на перевірку і за умови їх коректності, Controller далі передає ці аргументи у клас Model, що здійснює запит до бази даних.

Методи реалізовані до пункту 1 завдання лабораторної роботи:

DeleteFromTable – за умови правильності введених аргументів, наявності відповідного запису (з вказаним значенням первинного ключа) та незалежності інших таблиць від цього запису (до цього запису немає зовнішнього ключа з іншої таблиці), видаляє запис з вказаним первинним ключем.

UpdateTable – за умови правильності введених аргументів, наявності відповідного запису (з вказаним значенням первинного ключа) та записів інших таблиць (на які хочемо змінити поточні), змінює усі поля, окрім первинного ключа у обраному записі. Аргументи різні для кожної таблиці.

InsertInTable – за умови правильності введених аргументів, відсутності відповідного запису (з вказаним значенням первинного ключа) та наявності записів інших таблиць (на які хочемо посилатись зі створеного запису), вставляє новий рядок у таблицю з обраними значеннями полів. Аргументи різні для кожної таблиці.

Метод реалізований до пункту 2 завдання лабораторної роботи:

GenerateDataForTable – за умови введення правильного імені таблиці та числа n > 0, здійснює генерування n псевдорандомізованих записів у обраній таблиці. Аргументами є ім’я таблиці та число записів, що мають бути створені.

Метод реалізований до пункту 3 завдання лабораторної роботи:

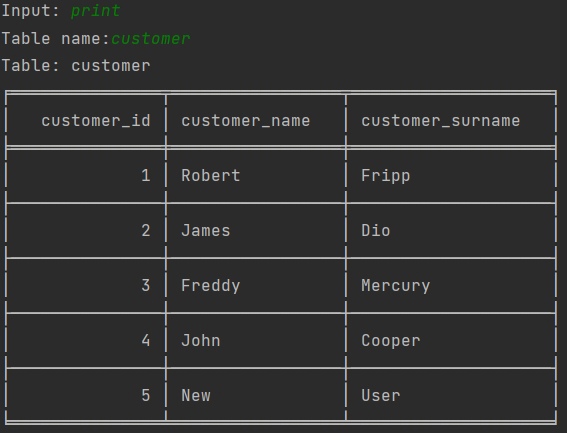
SearchForDataInTables() – за правильного задання умов пошуку, реалізує пошук за 1 та більше атрибутами з вказаних, і виводить у вікно терміналу результат пошуку (або нічого, якщо пошук не дав результатів) та час, за який було проведено запит. Не потребує початкових аргументі, користувач вибирає всередині методу, за якими ознаками шукати.

**Завдання 1**

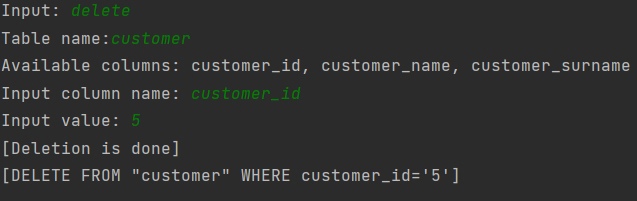
**Запит на видалення**

Для перевірки роботи розглянемо запити на видалення дочірньої таблиці discount\_card та батьківської таблиці customer.

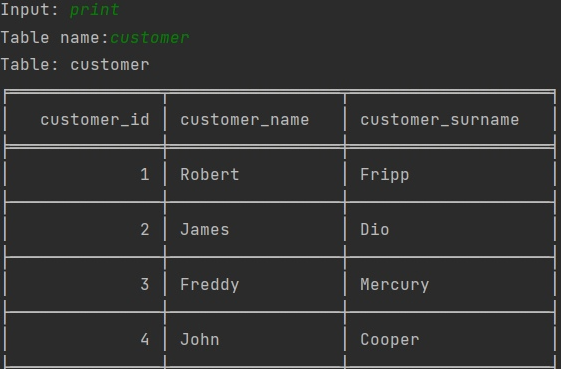
Таблиця customer до видалення запису 5:



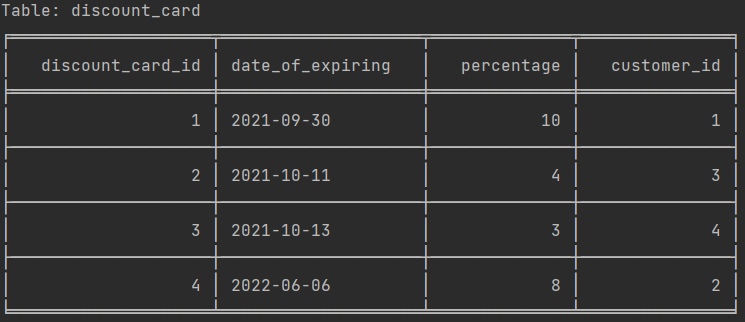
Запит на видалення:



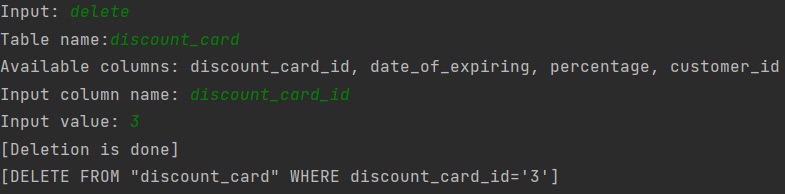
Таблиця customer після видалення запису 5:



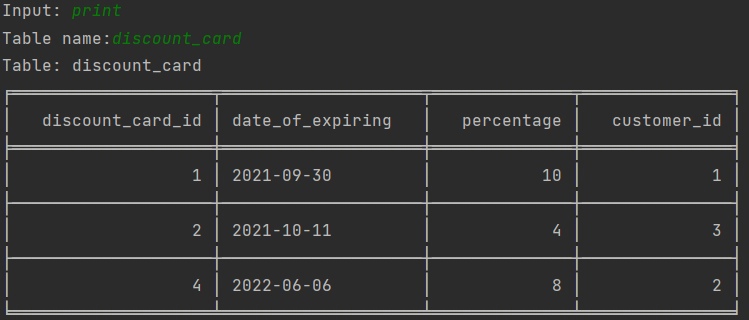
Таблиця discount\_card до видалення запису 3:



Запит на видалення:

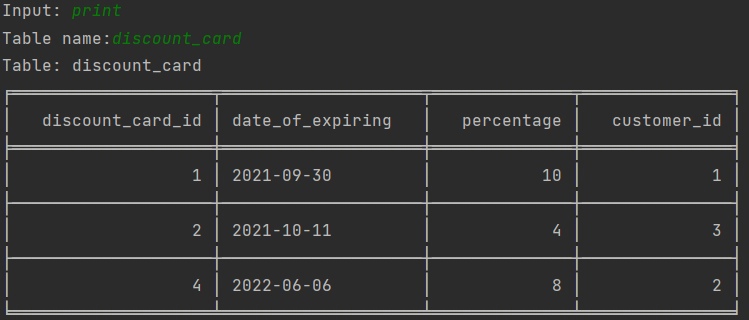


Таблиця discount\_card після видалення запису 3:

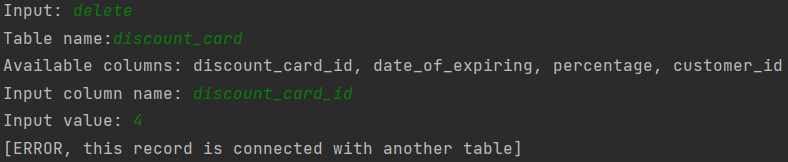


У даній програмній реалізації видалення запису з батьківської таблиці, який зв’язаний з дочірньою таблицею, не буде здійснено, а буде видано повідомлення про помилку.

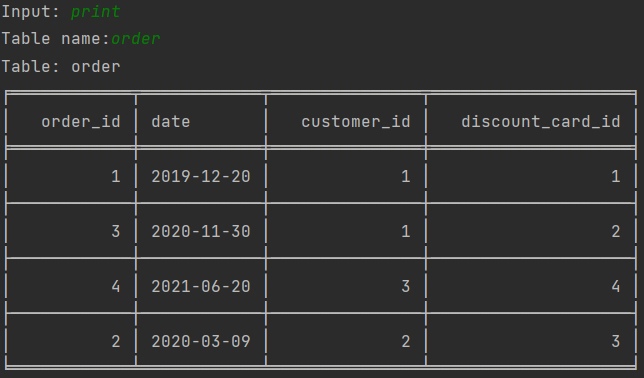
Таблиця discount\_card:



Запит на видалення та помилка:



Так як запис під номером 4 використовується в таблиці order, його неможливо видалити. Таблиця order:



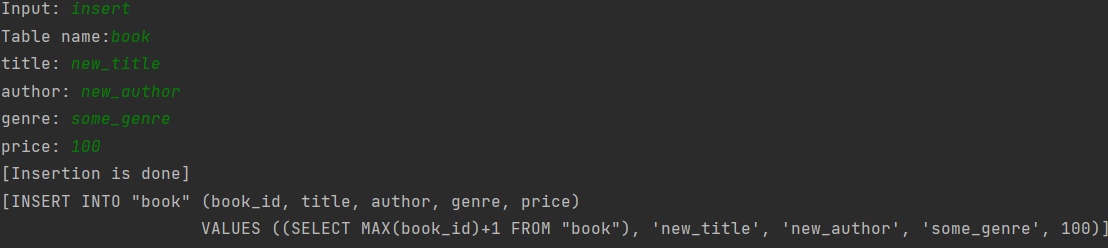
**Запит на вставку поля**

Для перевірки роботи розглянемо запити на видалення дочірньої таблиці book\_order та батьківської таблиці book.

Таблиця book до вставки нового поля:



Запит на додання нового поля в таблицю book з заданими атрибутами:

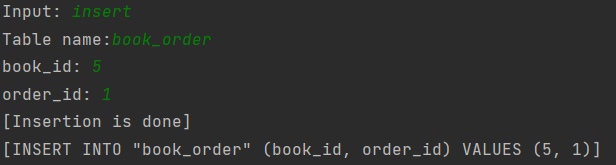


Таблиця book після успішного виконання запиту на вставку нового поля:

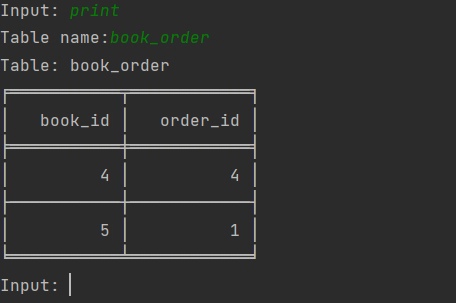


Таблиця book\_order до вставки нового поля:

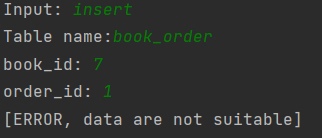


Запит на додання нового поля в таблицю book\_order з заданими атрибутами:

Таблиця book\_order після успішного виконання запиту на вставку нового поля:



Спроба вставки запису у дочірню таблицю book\_order з неіснуючим зовнішнім ключем 10:

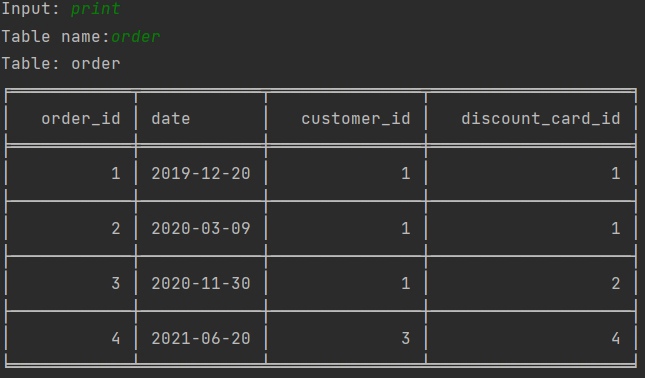


В таблиці book немає поля, в якому book\_id дорівнювало б 7, тому запит є невдалим.

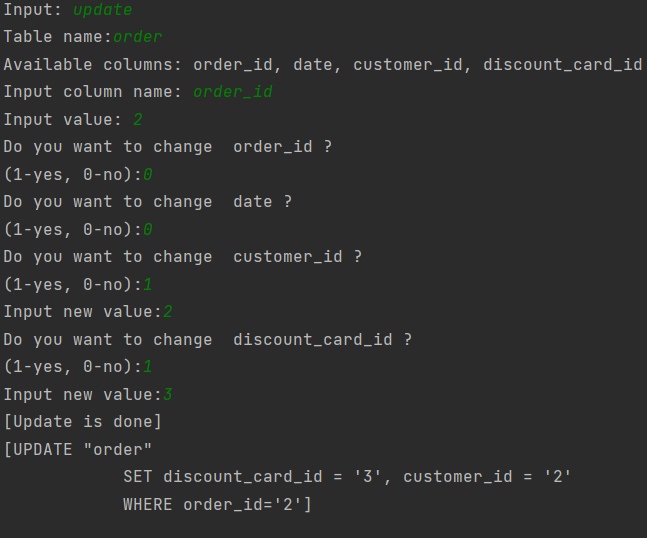
**Запит на зміну полів**

Для перевірки роботи розглянемо запити на зміну значенння в дочірній таблиці order. Спочатку коректний, потім з неіснуючим значенням зовнішнього ключа батьківської таблиці book.

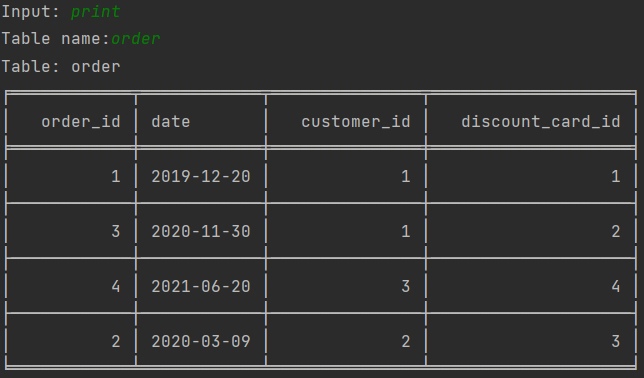
Таблиця order до запиту:



Запит на оновлення поля:



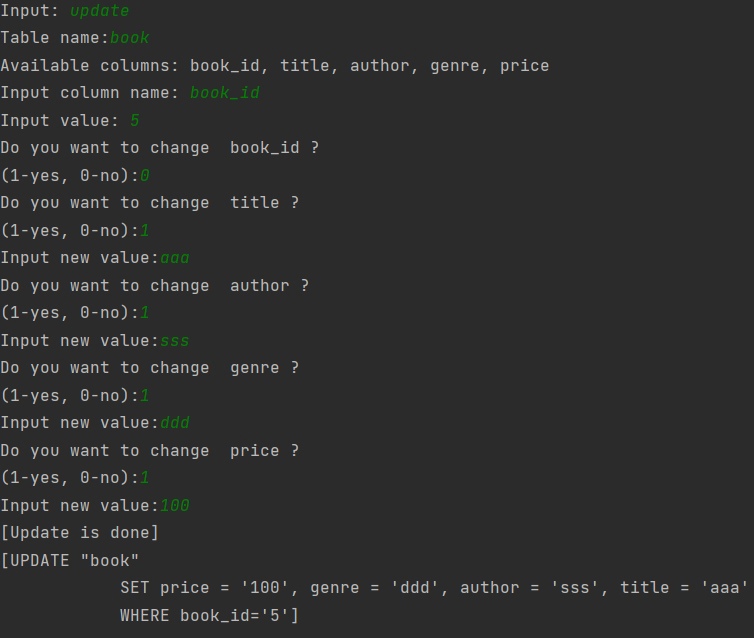
Таблиця order після запиту:



Таблиця book до запиту:



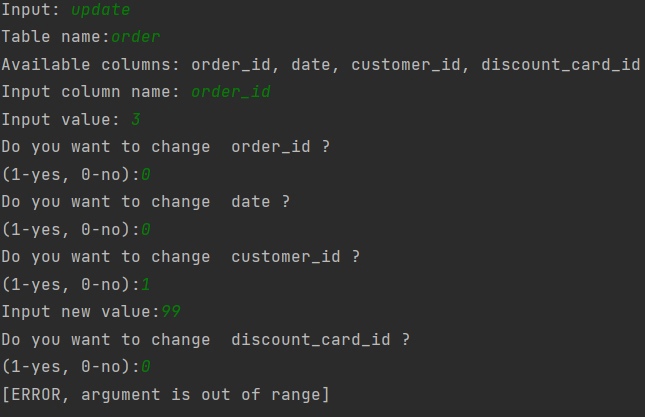
Запит на оновлення поля:



Таблиця book після запиту:



Спроба зміни запису у дочірній таблиці order з неіснуючим зовнішнім ключем 99:



Так як не існує поля customer\_id в таблиці customer, запит на зміну повертає помилку.

**Завдання 2**

Вставка псевдорандомізованих записів у кожну з таблиць

Таблиця book до запиту:



Запит на вставку 3 полів:

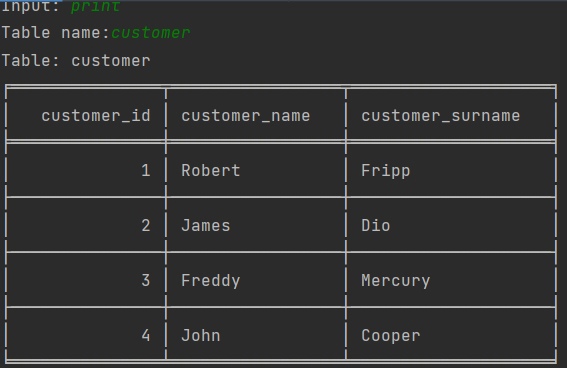
3 random fields were insert into book table

INSERT INTO "book" (book\_id, title, author, genre, price) VALUES ((SELECT MAX(book\_id)+1 FROM "book"), array\_to\_string(array(select chr((97 + round(random()\*25)) ::integer) FROM generate\_series(1,7)), ''), array\_to\_string(array(select chr((97 + round(random()\*25)) ::integer) FROM generate\_series(1,7)), ''), array\_to\_string(array(select chr((97 + round(random()\*25)) ::integer) FROM generate\_series(1,7)), ''), 1 + random()\*1000);

Таблиця book після запиту:



Таблиця сustomer до запиту:

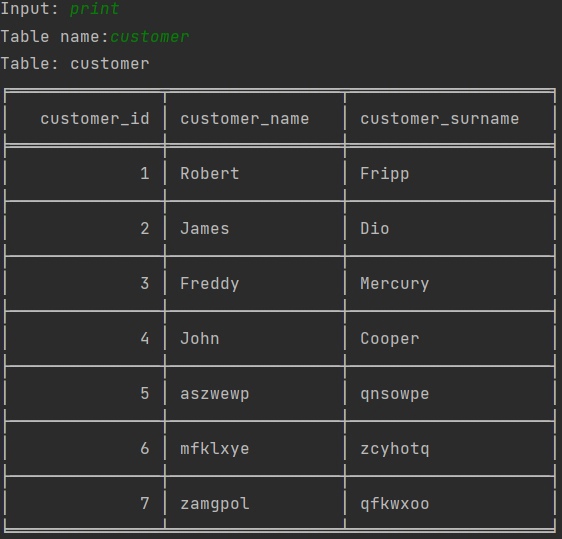


Запит на вставку 3 полів:

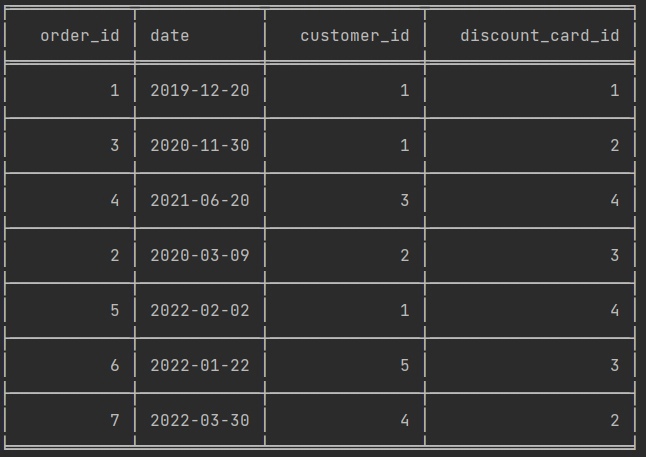
3 random fields were insert into customer table

INSERT INTO "customer" (customer\_id, customer\_name, customer\_surname) VALUES ((SELECT MAX(customer\_id)+1 FROM "customer"), array\_to\_string(array(select chr((97 + round(random()\*25)) ::integer) FROM generate\_series(1,7)), ''), array\_to\_string(array(select chr((97 + round(random()\*25)) ::integer) FROM generate\_series(1,7)), ''));

Таблиця customer після запиту:



Таблиця order до запиту:

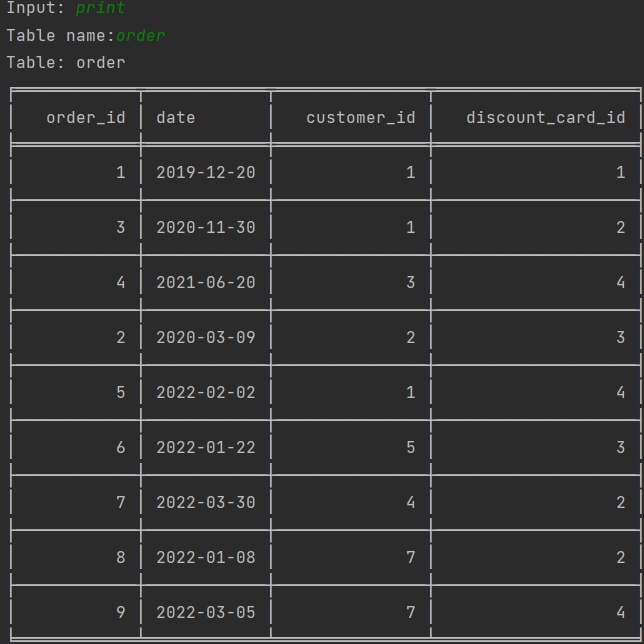


Запит на вставку 2 полів:

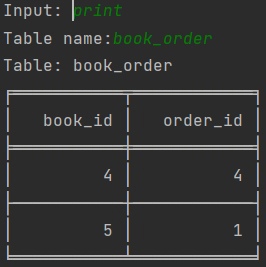
2 random fields were insert into order table

INSERT INTO "order" (order\_id, date, customer\_id, discount\_card\_id) VALUES ((SELECT MAX(order\_id)+1 FROM "order"), (SELECT NOW() + (random()\*(NOW()+'90 days' - NOW())) + '30 days'), (select (1 + random()\*(select max(customer\_id) FROM "customer"))::int), (select (1 + random()\*(select max(discount\_card\_id) FROM "discount\_card"))::int));

Таблиця order після запит**у**:



Таблиця book\_order до запиту:

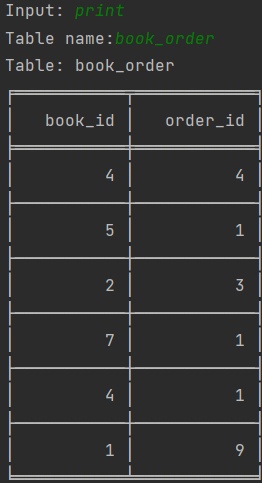


Запит на вставку 4 полів:

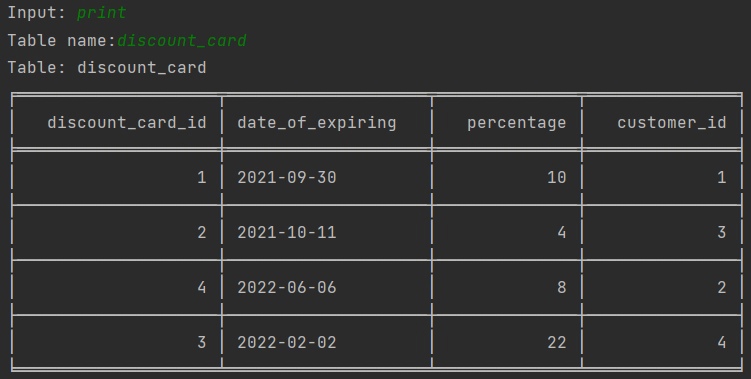
4 random fields were insert into book\_order table

INSERT INTO "book\_order" (book\_id, order\_id) VALUES ((select (1 + random()\*(select max(book\_id) FROM "book"))::int), (select (1 + random()\*(select max(order\_id) FROM "order"))::int));

Таблиця book\_order після запиту:



Таблиця discount\_card до запиту:

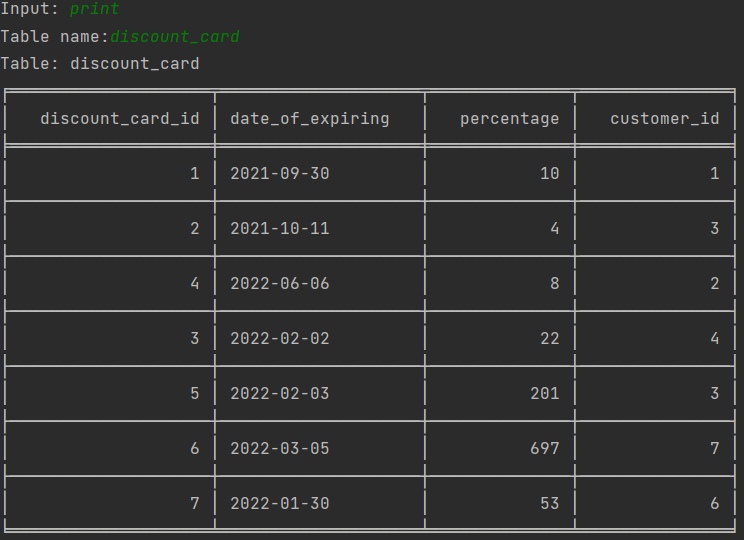


Запит на вставку 3 полів:

3 random fields were insert into discount\_card table

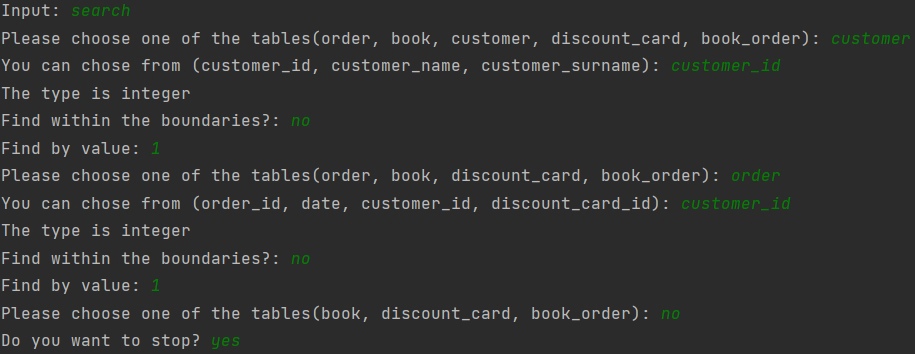
INSERT INTO "discount\_card" (discount\_card\_id, date\_of\_expiring, percentage, customer\_id) VALUES ((SELECT MAX(discount\_card\_id)+1 FROM "discount\_card"), (SELECT NOW() + (random()\*(NOW()+'90 days' - NOW())) + '30 days'), 1 + random()\*1000, (select (1 + random()\*(select max(customer\_id) FROM "customer"))::int));

Таблиця discount\_card після запиту:



**Завдання 3**

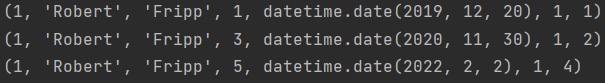
Пошук за одним атрибутом з двох таблиць customer та order



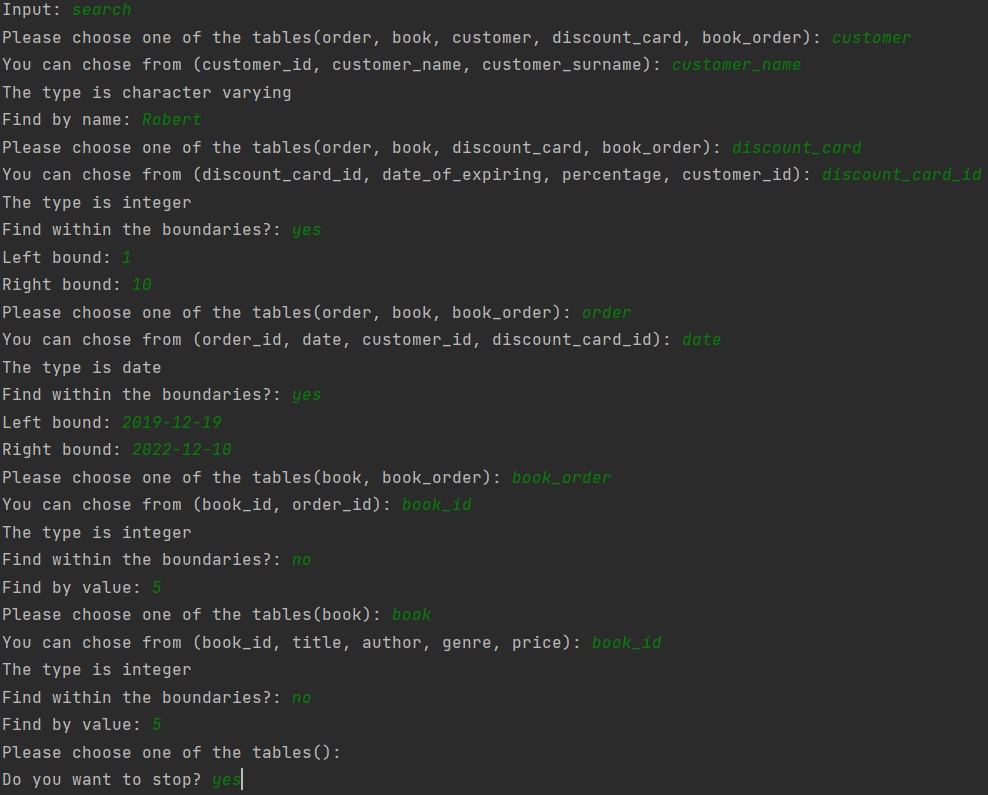
Запит:

SELECT \* from "customer" as one inner join "order" as two on one."customer\_id"=two."customer\_id" WHERE one.customer\_id=1 and two.customer\_id=1

Результат:



Пошук за чотирьма атрибутами з усіх п’яти таблиць



Запит:

SELECT \* from "customer" as one inner join "discount\_card" as two on one."customer\_id"=two."customer\_id" inner join "order" as three on one."customer\_id"=three."customer\_id" and two."discount\_card\_id"=three."discount\_card\_id" and two."customer\_id"=three."customer\_id" inner join "book\_order" as four on three."order\_id"=four."order\_id" inner join "book" as five on four."book\_id"=five."book\_id" WHERE one.customer\_name LIKE 'Robert' and 1<=two.discount\_card\_id and two.discount\_card\_id<=10 and three.date BETWEEN '2019-12-19'and '2022-12-10' and four.book\_id=5 and five.book\_id=5

Результат:

(1, 'Robert', 'Fripp', 1, datetime.date(2021, 9, 30), 10, 1, 1, datetime.date(2019, 12, 20), 1, 1, 5, 1, 5, 'aaa', 'sss', 'ddd', 100)

**Завдання 4**

**model.py**

import Connect

from View import View

class Model:

@staticmethod

def GetTables():

connection = Connect.MakeConnection()

cursor = connection.cursor()

cursor.execute("""SELECT table\_name

FROM information\_schema.tables

WHERE table\_schema = 'public'""")

tables = cursor.fetchall()

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

tables[i] = tables[i][0]

return tables

@staticmethod

def GetFields(table):

connection = Connect.MakeConnection()

cursor = connection.cursor()

cursor.execute("""SELECT column\_name

from information\_schema.columns

where table\_name = '{}'

order by ordinal\_position""".format(table))

names = cursor.fetchall()

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

names[i] = names[i][0]

cursor.close()

Connect.CloseConnection(connection)

return names

@staticmethod

def IfInDatabase(name):

if name in Model.GetTables():

return True

else:

return False

@staticmethod

def ShowTable(table):

connection = Connect.MakeConnection()

cursor = connection.cursor()

cursor.execute("""SELECT \* FROM public.{};""".format(table))

viewObj = View(table, Model.GetFields(table), cursor.fetchall())

viewObj.Show()

cursor.close()

Connect.CloseConnection(connection)

@staticmethod

def ShowAllTables():

connection = Connect.MakeConnection()

cursor = connection.cursor()

cursor.execute("""SELECT table\_name FROM information\_schema.tables WHERE table\_schema = 'public'""")

for table in cursor.fetchall():

Model.ShowTable(table[0])

cursor.close()

Connect.CloseConnection(connection)

@staticmethod

def InsertInTable(table):

connection = Connect.MakeConnection()

cursor = connection.cursor()

try:

opperation = View.InsertionView(table)

cursor.execute(opperation)

print(f"[Insertion is done]\n[{opperation}]\n")

except Exception as error:

print("[ERROR, data are not suitable]\n")

cursor.close()

Connect.CloseConnection(connection)

@staticmethod

def DeleteFromTable(table):

fields = Model.GetFields(table)

result = View.DeletionView(fields)

connection = Connect.MakeConnection()

cursor = connection.cursor()

try:

opperation = """DELETE FROM "{}" WHERE {}='{}'""".format(table, result[0], result[0])

cursor.execute(opperation)

print(f"[Deletion is done]\n[{opperation}]\n")

except Exception as error:

print("[ERROR, this record is connected with another table]\n")

cursor.close()

Connect.CloseConnection(connection)

@staticmethod

def UpdateTable(table):

fields = Model.GetFields(table)

res = View.UpdateView(fields)

connection = Connect.MakeConnection()

cursor = connection.cursor()

try:

result = """UPDATE "{}"

SET {}

WHERE {}='{}'""".format(table, res[0], res[1], res[2])

cursor.execute(result)

print(f"[Update is done]\n[{result}]\n")

except Exception as error:

print("[ERROR, argument is out of range]\n")

cursor.close()

Connect.CloseConnection(connection)

@staticmethod

def IsIdentity(table, column, cursor):

cursor.execute(

"""select is\_identity

from information\_schema.columns

where table\_name = '{}' and

column\_name = '{}'""".format(table, column))

return cursor.fetchone()[0] == 'NO'

@staticmethod

def GetColumnType(table, column):

connection = Connect.MakeConnection()

cursor = connection.cursor()

cursor.execute(

"""select data\_type

from information\_schema.columns

where table\_name = '{}' and

column\_name = '{}'""".format(table, column))

res = cursor.fetchone()[0]

cursor.close()

Connect.CloseConnection(connection)

return res

@staticmethod

def GenerateDataForTable(table, times):

fields = Model.GetFields(table)

connection = Connect.MakeConnection()

cursor = connection.cursor()

fieldsTypes = []

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

if Model.IsIdentity(table, fields[i], cursor):

type = Model.GetColumnType(table, fields[i])

result = View.CheckIfFk(table, fields[i])

if result is not None:

fieldsTypes.append(View.FindType(result))

else:

fieldsTypes.append(View.FindType(type))

else:

fieldsTypes.append('(SELECT MAX(' + fields[i] + ')+1 FROM "' + table + '")')

try:

word = """INSERT INTO "{}" ({})

VALUES ({});""".format(table, ", ".join(fields), ", ".join(fieldsTypes))

for i in range(0, times):

cursor.execute(word)

print(str(times) + " random fields were insert into " + str(table) + " table")

print(word)

except Exception as error:

print("[ERROR, something went wrong]", error)

cursor.close()

Connect.CloseConnection(connection)

@staticmethod

def FindSameColumns(first, second):

names = []

for i in Model.GetFields(first):

for j in Model.GetFields(second):

if i == j:

names.append(i)

return names

@staticmethod

def SearchForDataInTables():

names = ["one", "two", "three", "four", "five"]

tables = Model.GetTables()

data = []

while True:

print("Please choose one of the tables(" + ", ".join(tables) + "): ", end="")

tableInput = input()

if tableInput in tables:

tables.remove(tableInput)

columns = Model.GetFields(tableInput)

print("You can chose from (" + ", ".join(columns) + "): ", end="")

columnInput = input()

if columnInput in columns:

columnType = Model.GetColumnType(tableInput, columnInput)

print("The type is " + columnType)

if columnType == 'character varying':

inputValue = input("Find by name: ")

data.append([tableInput, columnInput, columnType, inputValue])

else:

inputValue = input("Find within the boundaries?: ")

if inputValue == 'yes':

leftBound = input("Left bound: ")

rightBound = input("Right bound: ")

data.append([tableInput, columnInput, columnType, [leftBound, rightBound]])

else:

inputValue = input("Find by value: ")

data.append([tableInput, columnInput, columnType, inputValue])

else:

cond = input("Do you want to stop? ")

if cond == 'yes' and len(tables) < 4:

break

upperWord = 'SELECT \* from "' + data[0][0] + '" as ' + names[0]

lowerWord = 'WHERE' + View.ChooseOperationByType(data[0][2], data[0][3], names[0], data[0][1])

for i in range(1, len(data)):

sameColumns = ""

upperWord = upperWord + ' inner join "' + data[i][0] + '" as ' + names[i]

for j in range(0, i):

res = Model.FindSameColumns(data[j][0], data[i][0])

for k in range(0, len(res)):

sameColumns = sameColumns + names[j] + '."' + res[k] + '"=' + names[i] + '."' + res[k] + '" and '

if len(sameColumns) > 1:

upperWord = upperWord + " on " + sameColumns[:-4]

lowerWord = lowerWord + View.ChooseOperationByType(data[i][2], data[i][3], names[i], data[i][1])

connection = Connect.MakeConnection()

cursor = connection.cursor()

try:

cursor.execute(upperWord + lowerWord[:-4])

info = cursor.fetchall()

for i in info:

print(i)

except Exception as error:

print("[ERROR, something went wrong]: ")

cursor.close()

Connect.CloseConnection(connection)

**view.py**

import Connect

from tabulate import tabulate

class View:

def \_\_init\_\_(self, table, fields, values):

self.table = table

self.fields = fields

self.values = values

def Show(self):

print('Table:', self.table)

print(tabulate(self.values, self.fields, tablefmt="fancy\_grid"))

@staticmethod

def GetInstruction():

print("all - Виводить всі таблиці, які є в базі даних\n"

"print - Виводить задану за іменем таблицю з бази даних\n"

"insert - Вставляє новий запис користувача в вибрану таблицю з бази даних\n"

"delete - Видаляє рядок з таблиці за заданим індексом\n"

"update - Оновлює вибрані значення певного рядка таблиці, вибрані користувачем\n"

"random - Заповнює вибрану таблицю вибраною кількістю згенерованих рядків\n"

"search - Реалізовю пошук за заданими атрибутами\n"

"exit - Вихід з програми\n")

@staticmethod

def UpdateView(fields):

print("Available columns:", end=" ")

print(', '.join(fields))

column = input("Input column name: ")

element = input("Input value: ")

valueString = ""

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

print("Do you want to change ", fields[i], "?")

condition = input("(1-yes, 0-no):")

if condition == '1':

newValue = input("Input new value:")

valueString = fields[i] + " = '" + newValue + "', " + valueString

return [valueString[:-2], column, element]

@staticmethod

def DeletionView(fields):

print("Available columns:", end=" ")

print(', '.join(fields))

column = input("Input column name: ")

element = input("Input value: ")

return [column, element]

@staticmethod

def InsertionView(table):

if table == 'book\_order':

book\_id = input("book\_id: ")

order\_id = input("order\_id: ")

return """INSERT INTO "book\_order" (book\_id, order\_id) VALUES ({}, {})""".format(book\_id, order\_id)

elif table == 'book':

title = input("title: ")

author = input("author: ")

genre = input("genre: ")

price = input("price: ")

return """INSERT INTO "book" (book\_id, title, author, genre, price)

VALUES ((SELECT MAX(book\_id)+1 FROM "book"), '{}', '{}', '{}', {})""".format(title, author, genre,

price)

elif table == 'discount\_card':

date\_of\_expiring = input("date of expiring: ")

percentage = input("percentage: ")

customer\_id = input("customer\_id: ")

return """INSERT INTO "discount\_card" (discount\_card\_id, date\_of\_expiring, percentage, customer\_id)

VALUES ((SELECT MAX(discount\_card\_id)+1 FROM "discount\_card"), '{}', {}, {})""".format(date\_of\_expiring,

percentage,

customer\_id)

elif table == 'order':

date = input("date: ")

discount\_card\_id = input("discount\_card\_id: ")

customer\_id = input("customer\_id: ")

return """INSERT INTO "order" (order\_id, date, customer\_id, discount\_card\_id)

VALUES ((SELECT MAX(order\_id)+1 FROM "order"), '{}', {}, {})""".format(date, customer\_id,

discount\_card\_id)

elif table == "customer":

customer\_name = input("customer\_name: ")

customer\_surname = input("customer\_surname: ")

return """INSERT INTO "customer" (customer\_id, customer\_name, customer\_surname)

VALUES ((SELECT MAX(customer\_id)+1 FROM "customer"), '{}', '{}')""".format(customer\_name,

customer\_surname)

@staticmethod

def CheckIfFk(table, column):

if column == 'customer\_id' and (table == "discount\_card" or table == "order"):

tableName = "customer"

ColumnName = "customer\_id"

elif column == 'discount\_card\_id' and table == "order":

tableName = "discount\_card"

ColumnName = "discount\_card\_id"

elif column == 'order\_id' and table == "book\_order":

tableName = "order"

ColumnName = "order\_id"

elif column == 'book\_id' and table == "book\_order":

tableName = "book"

ColumnName = "book\_id"

else:

return

return [tableName, ColumnName]

@staticmethod

def ChooseOperationByType(objType, objValue, objName, objColumn):

if objType == 'integer':

if type(objValue) is list:

return " {}<={}.{} and {}.{}<={} and".format(objValue[0], objName, objColumn, objName, objColumn, objValue[1])

else:

return " " + objName + "." + objColumn + "=" + objValue + " and"

elif objType == 'character varying':

return " " + objName + "." + objColumn + " LIKE '" + objValue + "' and"

else:

if type(objValue) is list:

return" {}.{} BETWEEN '{}'and '{}' and".format(objName, objColumn, objValue[0], objValue[1])

else:

return " " + objName + "." + objColumn + "='" + objValue + "' and"

@staticmethod

def FindType(typeName):

if typeName == 'date':

return View.DateType()

elif typeName == 'character varying':

return View.StringType()

elif typeName == 'integer':

return View.IntegerType()

else:

return View.ForeignKeyIntegerType(typeName)

@staticmethod

def ForeignKeyIntegerType(value):

return '(select (1 + random()\*(select max(' + value[1] +') FROM "' + value[0] + '"))::int)'

@staticmethod

def IntegerType():

return "1 + random()\*1000"

@staticmethod

def StringType():

return "array\_to\_string(array(" \

"select chr((" \

"97 + round(random()\*25)) ::integer) " \

"FROM generate\_series(1,7)), '')"

@staticmethod

def DateType():

return "(SELECT NOW() + " \

"(random()\*(NOW()+'90 days' - NOW())) " \

"+ '30 days')"