

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»**

Тема: «Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL»

Виконала: студентка ІII курсу

ФПМ групи КВ-83

Плюхович Надія Івванівна

Перевірив: Павловський В.І.

Київ – 2020

# Завдання

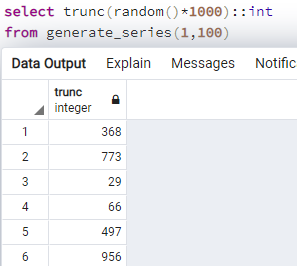
*Загальне завдання* роботи полягає у наступному:

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).

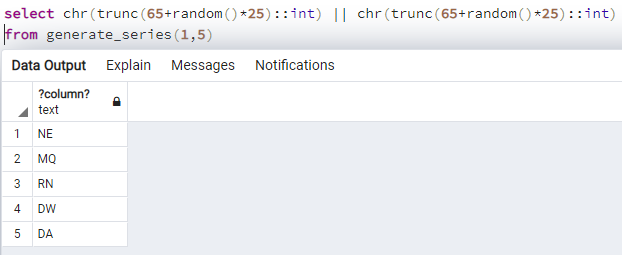
*Деталізоване завдання*:

1. 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.
2. 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**!

Приклад генерації 100 псевдовипадкових чисел:



Приклад генерації 5 псевдовипадкових рядків:



Приклад генерації псевдовипадкової мітки часу з діапазону [доступний за посиланням](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061).

Кількість даних для генерування має вводити користувач з клавіатури. Для тесту взяти 100 000 записів для однієї-двох таблиць.

Особливу увагу слід звернути на відповідність даних вимогам зовнішніх ключів з метою уникнення помилок порушення обмежень цілісності (foreign key).

1. 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.
2. Програмний код організувати згідно шаблону Model-View-Controller(MVC).  Приклад організації коду згідно шаблону доступний [за даним посиланням](https://www.giacomodebidda.com/mvc-pattern-in-python-introduction-and-basicmodel/). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

Рекомендована бібліотека взаємодії з PostgreSQL Psycopg2: http://initd.org/psycopg/docs/usage.html)

*Вимоги до інтерфейсу користувача*

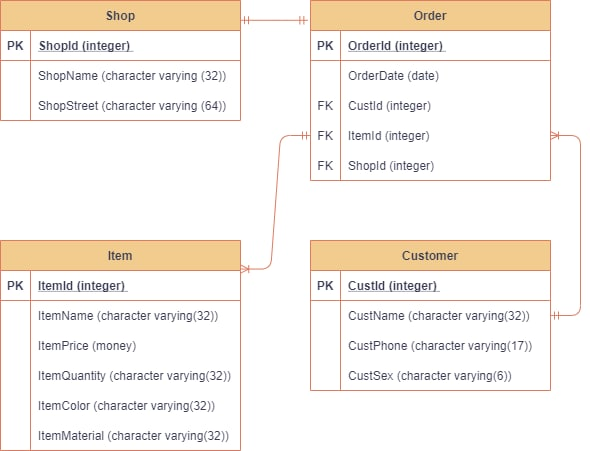
Використовувати консольний інтерфейс користувача.

*Вимоги до інструментарію*

 Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

# Нормалізована модель даних БД «Магазини»

# На рисунку зображена нормалізована модель даних БД ‘Тестування вакцини’, розроблена на першій лабораторній роботі.



Усі зв’язки між таблицями, типи атрибутів таблиць та ключі описані нижче.

**Опис структури БД “Магазини”**

|  |  |  |
| --- | --- | --- |
| Відношення | Атрибут | Тип |
| Відношення „Shop“ вміщує інформацію про магзин | ShopId – унікальний ID магазину. UNIQUE  ShopName – назва магазину. Не допускає NULL.  ShopStreet – назва вулиці, де знаходиться магазин. Не допускає NULL. | integer(PK)  character varying(32)  character varying(64) |
| Відношення “Item” вміщує інформацію про товар | ItemId – унікальний ID товару. UNIQUE.  ItemName – назва товару. НЕ допускає NULL.  ItemPrice – ціна товару. Не допускає NULL.  ItemQuantity – кількість. Не допускає NULL.  ItemColor – колір товару  ItemMaterial – матеріал товару. Не допускає NULL | integer(PK)  character varying(32)  money  character varying(32)  character varying(32)  character varying(32) |
| Відношення “Customer” вміщує інформацію про клієнт | CustId – унікальний ID покупця. UNIQUE.  CustName – ім’я покупця. Не допускає NULL.  CustPhone – телефон покупця.UNIQUE.  CustSex – стать покупця. Не допускає NULL. | integer(PK)  character varying(32)  character varying(17)  character varying(6) |
| Відношення „Order“ вміщує інформацію про товар | OrderId – унікальний ID заказу.UNIQUE  OrderDate – дата зробленого заказу. Не допускає NULL.  CustId – ID покупця, що робить заказ. Не допускає NULL.  ItemId – ID товару, який продавець збирається покупати. Не допускає NULL.  ShopId – ID магазину, де покупець покупає товар. Не допускає NULL. | integer(PK)  date  integer(FK)  integer(FK)  integer(FK) |

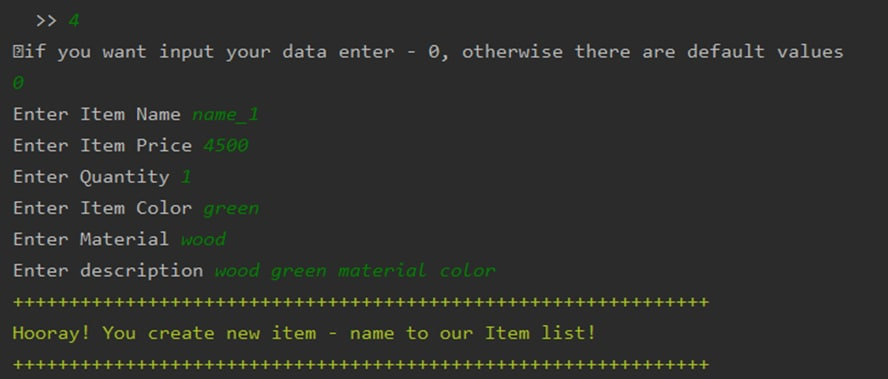
**Опис програми**

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL і мови програмування Python, та реалізовує функціональні вимоги, що наведені у завданні. Додаток використовує шаблон проектування MVC.

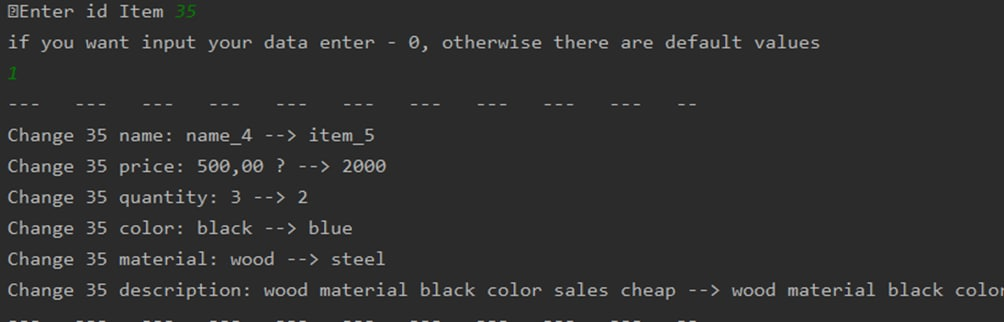
# Завдання №1

Приклад роботи програми, що виконує вставку, оновлення та видалення даних. При успішному виконанні команди програма сповіщає про це у вигляді повідомлення.

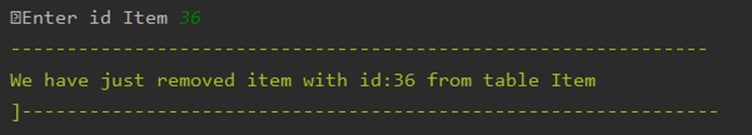
Вставка



Оновлення



Видалення



**Код функцій для основних операцій, що використовує модель:**

def insert\_one\_shop(connection, name, street):

try:

cursor = connection.cursor()

postgres\_insert\_query = """ INSERT INTO public.\"Shop\" (\"ShopName\", \"ShopStreet\")

VALUES (%s,%s)"""

record\_to\_insert = (name, street)

cursor.execute(postgres\_insert\_query, record\_to\_insert)

connection.commit()

cursor.close()

except (Exception, psycopg2.Error) as err:

if connection:

print("Failed to insert record into table", err)

def insert\_many\_random\_shops(connection, count):

try:

cursor = connection.cursor()

cursor.execute("INSERT INTO public.\"Shop\"(\"ShopName\", \"ShopStreet\") select left(md5(random()::text), 10),"

"left(md5(random()::text), 10) from generate\_series(1, {})".format(count))

connection.commit()

cursor.close()

except (Exception, psycopg2.Error) as err:

if connection:

print("Failed to insert record into table", err)

def update\_one\_shop(connection, shopId, name, street):

try:

cursor = connection.cursor()

sql\_update\_query = """UPDATE public.\"Shop\" SET \"ShopName\"='{}', \"ShopStreet\"='{}' WHERE \"ShopId\"={}""" \

.format(name, street, shopId)

cursor.execute(sql\_update\_query)

connection.commit()

except (Exception, psycopg2.Error) as err:

print("Error while updating PostgresSQL table", err)

def insert\_one\_customer(connection, name, phone, sex):

try:

cursor = connection.cursor()

postgres\_insert\_query = """ INSERT INTO public.\"Customer\" (\"CustomerName\", \"CustomerPhone\", \"CustomerSex\")

VALUES (%s,%s,%s)"""

record\_to\_insert = (name, phone, sex)

cursor.execute(postgres\_insert\_query, record\_to\_insert)

connection.commit()

cursor.close()

except (Exception, psycopg2.Error) as err:

if connection:

print("Failed to insert record into table", err)

def insert\_many\_random\_customers(connection, count):

try:

cursor = connection.cursor()

cursor.execute(

"INSERT INTO public.\"Customer\"(\"CustomerName\", \"CustomerPhone\", \"CustomerSex\") select left(md5(random()::text), 10),"

" format('+380 %s%s %s%s%s %s%s %s%s', a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8], a[9]),"

" ('male') from (select ARRAY (SELECT trunc(random() \* 10)::int FROM generate\_series(1, 9)) AS a,generate\_series(1, {})) sub;".format(

count))

connection.commit()

cursor.close()

except (Exception, psycopg2.Error) as err:

if connection:

print("Failed to insert record into table", err)

def update\_one\_customer(connection, customerId, name, phone, sex):

try:

cursor = connection.cursor()

sql\_update\_query = """UPDATE public.\"Customer\" SET \"CustomerName\"='{}', \"CustomerPhone\"='{}', \"CustomerSex\"='{}'

WHERE \"CustomerId\"={}""" \

.format(name, phone, sex, customerId)

cursor.execute(sql\_update\_query)

connection.commit()

except (Exception, psycopg2.Error) as err:

print("Error while updating PostgresSQL table", err)

def insert\_one\_item(connection, name, price, quantity, color, material, description):

try:

cursor = connection.cursor()

postgres\_insert\_query = """ INSERT INTO public.\"Item\" (\"ItemName\", \"ItemPrice\", \"ItemQuantity\", \"ItemColor\", \"ItemMaterial\", \"ItemDescriptions\")

VALUES (%s,%s,%s,%s,%s,%s)"""

record\_to\_insert = (name, price, quantity, color, material, description)

cursor.execute(postgres\_insert\_query, record\_to\_insert)

connection.commit()

cursor.close()

except (Exception, psycopg2.Error) as err:

if connection:

print("Failed to insert record into table", err)

def insert\_many\_random\_items(connection, count):

try:

cursor = connection.cursor()

cursor.execute(

"INSERT INTO public.\"Item\"(\"ItemName\", \"ItemPrice\", \"ItemQuantity\", \"ItemColor\", \"ItemMaterial\", \"ItemDescriptions\") select left(md5(random()::text), 10), floor(random() \* 100 + 100)::int, floor(random() \* 10 + 1)::int, left(md5(random()::text), 10), left(md5(random()::text), 10), left(md5(random()::text), 100) from generate\_series(1, {})".format(

count))

connection.commit()

cursor.close()

except (Exception, psycopg2.Error) as err:

if connection:

print("Failed to insert record into table", err)

def update\_one\_item(connection, itemId, name, price, quantity, color, material, description):

try:

cursor = connection.cursor()

sql\_update\_query = """UPDATE public.\"Item\" SET \"ItemName\"='{}', \"ItemPrice\"='{}', \"ItemQuantity\"='{}', \"ItemColor\"='{}', \"ItemMaterial\"='{}', \"ItemDescriptions\"='{}'

WHERE \"ItemId\"={}""" \

.format(name, price, quantity, color, material, description, itemId)

cursor.execute(sql\_update\_query)

connection.commit()

except (Exception, psycopg2.Error) as err:

print("Error while updating PostgresSQL table", err)

def insert\_one\_order(connection, itemId, shopId, customerId, date):

try:

cursor = connection.cursor()

postgres\_insert\_query = """ INSERT INTO public.\"Order\" (\"ItemId\", \"ShopId\", \"CustomerId\", \"OrderDate\")

VALUES (%s,%s,%s,%s)"""

record\_to\_insert = (itemId, shopId, customerId, date)

cursor.execute(postgres\_insert\_query, record\_to\_insert)

connection.commit()

cursor.close()

except (Exception, psycopg2.Error) as err:

if connection:

print("Failed to insert record into table", err)

def insert\_many\_random\_orders(connection, count, itemId, customerId, shopId):

try:

cursor = connection.cursor()

cursor.execute(

"INSERT INTO public.\"Order\"(\"CustomerId\", \"ItemId\", \"ShopId\", orderdate) select {}, {}, {}, "

"date((current\_date - '15 years'::interval) + trunc(random() \* 365) \* '1 day'::interval + "

"trunc(random() \* 14) \* '1 year'::interval ) from generate\_series(1, {})".format(

customerId, itemId, shopId, count))

connection.commit()

cursor.close()

except (Exception, psycopg2.Error) as err:

if connection:

print("Failed to insert record into table", err)

def update\_one\_order(connection, orderId, itemId, shopId, customerId, date):

try:

cursor = connection.cursor()

sql\_update\_query = """UPDATE public.\"Order\" SET \"ItemId\"='{}', \"ShopId\"='{}', \"CustomerId\"='{}', \"OrderDate\"='{}'

WHERE \"OrderId\"={}""" \

.format(itemId, shopId, customerId, date, orderId)

cursor.execute(sql\_update\_query)

connection.commit()

except (Exception, psycopg2.Error) as err:

print("Error while updating PostgresSQL table", err)

def delete\_one(connection, anyId, nameTable):

try:

cursor = connection.cursor()

ps\_delete\_query = """DELETE FROM public.\"{}\" WHERE \"{}Id\"={};""".format(nameTable, nameTable, anyId)

cursor.execute(ps\_delete\_query)

connection.commit()

except (Exception, psycopg2.Error) as err:

print("Error while connecting to PostgresSQL", err)

def select\_all(connection, tableName):

try:

cursor = connection.cursor()

postgreSQL\_select\_Query = "SELECT \* from \"{}\"".format(tableName)

cursor.execute(postgreSQL\_select\_Query)

records = cursor.fetchall()

return records

except (Exception, psycopg2.Error) as err:

print("Error fetching data from PostgresSQL table", err)

def select\_one(connection, tableName, itemId):

try:

cursor = connection.cursor()

postgreSQL\_select\_Query = "SELECT \* from \"{}\" WHERE \"{}Id\"={}".format(tableName, tableName, itemId)

cursor.execute(postgreSQL\_select\_Query)

record = cursor.fetchall()

return record

except (Exception, psycopg2.Error) as err:

print("Error fetching data from PostgresSQL table", err)

На рівні контролеру виконується виклик функцій моделі, виведення результатів роботи, або помилок. Контролер дозволяє обробку будь-яких виключень, що можуть виникнути під час роботи програми в тому числі і посилальної цілісністю двох пов'язаних таблиць. Далі наведемо фрагмент коду контролера, що забезпечує обробку і виведення помилок.

class ControllerShop(object):

def init(self, model, view):

self.model = model()

self.view = view()

def insert\_shop(self, shop):

try:

shop = Input.input\_update\_shop(shop)

assert Check.isExistAllOptionShop(shop), '\033[91m you don\'t add all needed shop\'s option \033[0m'

self.model.create\_shop(shop['name'], shop['street'])

self.view.display\_stored(shop['name'], self.model.tableName)

except Exception as err:

print(err)

def insert\_many\_shops(self):

try:

count = Input.input\_count()

assert count > 1, '\033[91m count must be more one \033[0m'

self.model.create\_many\_shops(count)

self.view.display\_many\_stored(count, self.model.tableName)

except Exception as err:

print(err)

def disconnect(self):

self.model.disconnect\_from\_db()

self.view.delete\_connection()

def show\_shops(self):

items = self.model.read\_shops()

self.view.show\_list\_shops(items)

def show\_shop(self):

shopId = Input.input\_id(self.model.tableName)

shop = self.model.read\_shop(shopId)

self.view.show\_list\_shops(shop)

def update\_shop(self, shop):

shopId = Input.input\_id(self.model.tableName)

shop = Input.input\_update\_shop(shop)

older = self.model.read\_shop(shopId)

newShop = Check.updateShop(shop, [older[0].ShopId, older[0].ShopName, older[0].ShopStreet])

check = self.model.update\_shop(shopId, newShop['name'], newShop['street'])

if check:

print(older[0].ShopName, older[0].ShopStreet)

self.view.display\_shop\_updated(shopId, newShop['name'], newShop['street'])

def delete\_shop(self):

shopId = Input.input\_id(self.model.tableName)

check = self.model.delete\_shop(shopId)

if check:

self.view.display\_deletion(shopId, self.model.tableName)

class ControllerCustomer(object):

def init(self, model, view):

self.model = model()

self.view = view()

def insert\_customer(self, customer):

try:

customer = Input.input\_update\_customer(customer)

assert Check.isExistAllOptionCustomer(customer), \

'\033[91m you don\'t add all needed customer\'s option \033[0m '

check = self.model.create\_customer(customer['name'], customer['phone'], customer['sex'])

if check:

self.view.display\_stored(customer['name'], self.model.tableName)

except Exception as err:

print(err)

def insert\_many\_customers(self):

try:

count = Input.input\_count()

assert count > 1, '\033[91m count must be more one \033[0m'

self.model.create\_many\_customers(count)

self.view.display\_many\_stored(count, self.model.tableName)

except Exception as err:

print(err)

def disconnect(self):

self.model.disconnect\_from\_db()

self.view.delete\_connection()

def show\_customers(self):

items = self.model.read\_customers()

self.view.show\_list\_customers(items)

def show\_customer(self):

customerId = Input.input\_id(self.model.tableName)

customer = self.model.read\_customer(customerId)

self.view.show\_list\_customers(customer)

def update\_customer(self, customer):

customerId = Input.input\_id(self.model.tableName)

customer = Input.input\_update\_customer(customer)

older = self.model.read\_customer(customerId)

newCustomer = Check.updateCustomer(customer, older)

check = self.model.update\_customer(customerId, newCustomer['name'], newCustomer['phone'], newCustomer['sex'])

if check:

self.view.display\_customer\_updated(customerId, newCustomer['name'], newCustomer['phone'],

newCustomer['sex'])

def delete\_customer(self):

customerId = Input.input\_id(self.model.tableName)

check = self.model.delete\_customer(customerId)

if check:

self.view.display\_deletion(customerId, self.model.tableName)

class ControllerItem(object):

def init(self, model, view):

self.model = model()

self.view = view()

def insert\_item(self, items):

try:

items = Input.input\_update\_item(items)

description = None

assert Check.isExistAllOptionItem(items), '\033[91m you don\'t add all needed item\'s option \033[0m '

if 'description' in items:

description = items['description']

check = self.model.create\_item(items['name'], items['price'], items['quantity'], items['color'],

items['material'], description)

if check:

self.view.display\_stored('name', self.model.tableName)

except Exception as err:

print(err)

def insert\_many\_items(self):

try:

count = Input.input\_count()

assert count > 1, '\033[91m count must be more one \033[0m'

self.model.create\_many\_items(count)

self.view.display\_many\_stored(count, self.model.tableName)

except Exception as err:

print(err)

def disconnect(self):

self.model.disconnect\_from\_db()

self.view.delete\_connection()

def show\_items(self):

items = self.model.read\_items()

self.view.show\_list\_items(items)

def show\_item(self):

itemId = Input.input\_id(self.model.tableName)

items = self.model.read\_item(itemId)

self.view.show\_list\_items(items)

def update\_item(self, item):

itemId = Input.input\_id(self.model.tableName)

item = Input.input\_update\_item(item)

older = self.model.read\_item(itemId)

item = Check.updateItem(item, [older[0].ItemId, older[0].ItemName, older[0].ItemPrice, older[0].ItemQuantity,

older[0].ItemColor, older[0].ItemMaterial, older[0].ItemDescriptions])

check = self.model.update\_item(itemId, item['name'], item['price'], item['quantity'],

item['color'], item['material'], item['description'])

if check:

self.view.display\_item\_updated(itemId, older, item['name'], item['price'], item['quantity'],

item['color'], item['material'], item['description'])

def delete\_item(self):

itemId = Input.input\_id(self.model.tableName)

check = self.model.delete\_item(itemId)

if check:

self.view.display\_deletion(itemId, self.model.tableName)

class ControllerOrder(object):

def init(self, model, view):

self.model = model()

self.view = view()

def insert\_order(self, order):

try:

order = Input.input\_update\_order(order)

assert Check.isExistAllOptionOrder(order), \

'\033[91m you don\'t add all needed order\'s option \033[0m'

check = self.model.create\_order(order['itemId'], order['shopId'], order['customerId'], order['date'])

if check:

self.view.display\_stored("", self.model.tableName)

except Exception as err:

print(err)

def insert\_many\_orders(self):

try:

count = Input.input\_count()

assert count > 1, '\033[91m count must be more one \033[0m'

check = self.model.create\_many\_orders(count)

if check:

self.view.display\_many\_stored(count, self.model.tableName)

except Exception as err:

print(err)

def disconnect(self):

self.model.disconnect\_from\_db()

self.view.delete\_connection()

def show\_orders(self):

orders = self.model.read\_orders()

self.view.show\_list\_orders(orders)

def show\_order\_by\_quantity(self):

try:

values = Input.input\_two\_value()

assert values[0] <= values[1], '\033[91m first value must be less than second value\033[0m'

items = self.model.read\_order\_another\_table(values[0], values[1])

self.view.show\_list\_items\_orders(items, "ItemQuantity |")

except Exception as err:

print(err)

return False

def show\_order\_by\_itemName(self):

name = Input.input\_name()

items = self.model.read\_order\_itemName(name)

self.view.show\_list\_items\_orders(items, "ItemName |")

def show\_order(self):

orderId = Input.input\_id(self.model.tableName)

order = self.model.read\_order(orderId)

self.view.show\_list\_orders(order)

def update\_order(self, order):

orderId = Input.input\_id(self.model.tableName)

order = Input.input\_update\_order(order)

older = self.model.read\_order(orderId)

newOrder = Check.updateOrder(order, [older[0].OrderId, older[0].CustomerId, older[0].ItemId,

older[0].ShopId, older[0].OrderDate])

print(newOrder)

check = self.model.update\_order(orderId, newOrder['itemId'], newOrder['shopId'], newOrder['customerId'],

newOrder['date'])

def delete\_order(self):

orderId = Input.input\_id(self.model.tableName)

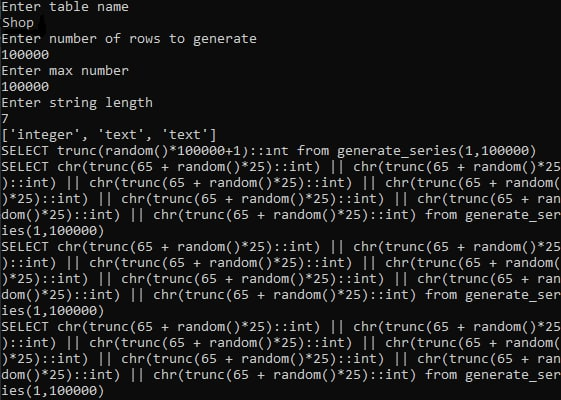
check = self.model.delete\_order(orderId)

if check:

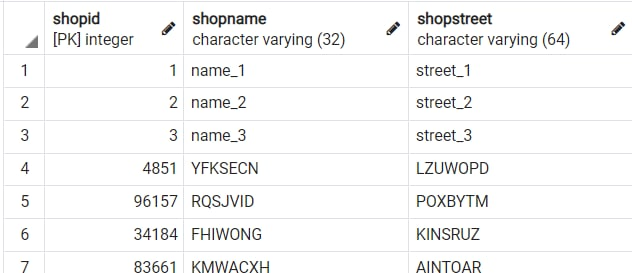
self.view.display\_deletion(orderId, self.model.tableName)

# Завдання №2

Приклад роботи генерації псевдовипадкових рядків в таблицях Shop:



Результат



Запити, що було створено для таблиці. Для кожного стовпця по одному запиту.

SELECT trunc(random()\*100000+1)::int from generate\_series(1,100000)

SELECT chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) from generate\_series(1,100000)

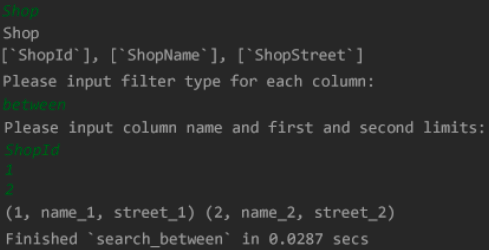
SELECT chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) || chr(trunc(65 + random()\*25)::int) from generate\_series(1,100000)

**Завдання 3**

Між певними значеннями

cur.execute(**"""SELECT \* FROM {}**

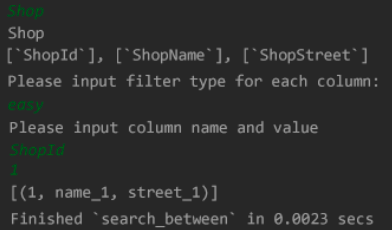
**WHERE {} BETWEEN {} AND {}"""**.format(table\_name, column\_name, limit1, limin2))



Звичайний easy пошук

cur.execute(**"""SELECT \* FROM {}**

**WHERE {} = '{}' ORDER BY {}"""**.format(table\_name, column\_name, value, column\_name))



# Код програми

Файл **controller.py** та **backendSQL.py** були наведені вище

Файл **model.py**

import psycopg2

from BD\_2 import backendSQL as baseData

from BD\_2.additional import Additional

class Model(object):

def init(self):

self.\_connection = self.connect\_to\_db()

@property

def connection(self):

return self.\_connection

@staticmethod

def connect\_to\_db():

connection = psycopg2.connect(user="postgres", password="password", host="127.0.0.1", port="5432",

database="Shops")

return connection

def disconnect\_from\_db(self):

if self.connection is not None:

self.connection.close()

class ModelShop(Model):

def init(self):

super().init()

self.\_tableName = "Shop"

@property

def tableName(self):

return self.\_tableName

def create\_shop(self, name, street):

baseData.insert\_one\_shop(self.connection, name, street)

def create\_many\_shops(self, count):

baseData.insert\_many\_random\_shops(self.connection, count)

def update\_shop(self, shopId, name, street):

try:

assert Additional.findExistingId(self.connection, self.tableName, shopId), \

'\033[91m id isn\'t exist \033[0m '

baseData.update\_one\_shop(

self.connection, shopId, name, street)

return True

except Exception as err:

print(err)

return False

def read\_shops(self):

return baseData.select\_all(self.connection, self.\_tableName)

def read\_shop(self, shopId):

return baseData.select\_one(self.connection, self.\_tableName, shopId)

def delete\_shop(self, shopId):

try:

assert not Additional.findExistingIdOrderTable(self.connection, self.tableName, shopId), \

'\033[91m shop id bind with order \033[0m'

assert Additional.findExistingId(self.connection, self.tableName, shopId), \

'\033[91m id isn\'t exist \033[0m'

baseData.delete\_one(self.connection, shopId, self.tableName)

return True

except Exception as err:

print(err)

return False

class ModelCustomer(Model):

def init(self):

super().init()

self.\_tableName = "Customer"

@property

def tableName(self):

return self.\_tableName

def create\_customer(self, name, phone, sex):

try:

assert (sex == "male" or sex == "female"), \

'\033[91m sex isn\'t exist. You must input male or female \033[0m'

baseData.insert\_one\_customer(self.connection, name, phone, sex)

return True

except Exception as err:

print(err)

return False

def create\_many\_customers(self, count):

baseData.insert\_many\_random\_customers(self.connection, count)

def update\_customer(self, customerId, name, phone, sex):

try:

assert (sex == "male" or sex == "female"), \

'\033[91m sex isn\'t exist. You must input male or female \033[0m'

assert Additional.findExistingId(self.connection, self.tableName, customerId), \

'\033[91m id isn\'t exist \033[0m '

baseData.update\_one\_customer(

self.connection, customerId, name, phone, sex)

return True

except Exception as err:

print(err)

return False

def read\_customers(self):

return baseData.select\_all(self.connection, self.\_tableName)

def read\_customer(self, customerId):

return baseData.select\_one(self.connection, self.\_tableName, customerId)

def delete\_customer(self, customerId):

try:

assert not Additional.findExistingIdOrderTable(self.connection, self.tableName, customerId),\

'\033[91m You don\'t delete customer, because this customer bind with order \033[0m'

assert Additional.findExistingId(self.connection, self.tableName, customerId), \

'\033[91m id isn\'t exist \033[0m'

baseData.delete\_one(self.connection, customerId, self.tableName)

return True

except Exception as err:

print(err)

return False

class ModelItem(Model):

def init(self):

super().init()

self.\_tableName = "Item"

@property

def tableName(self):

return self.\_tableName

def create\_item(self, name, price, quantity, color, material, description):

try:

assert price > 0, '\033[91m price must be more zero \033[0m'

assert quantity > 0, '\033[91m quantity must be > 0 \033[0m '

baseData.insert\_one\_item(self.connection, name, price, quantity, color, material, description)

return True

except Exception as err:

print(err)

return False

def create\_many\_items(self, count):

baseData.insert\_many\_random\_items(self.connection, count)

def read\_items(self, value, word=None):

if word and value == 1:

strWord = Additional.addLogicOperation(word)

return Additional.findWordInText(self.connection, strWord)

elif value == 2:

return Additional.findTextWithoutWord(self.connection, word)

return baseData.select\_all(self.connection, self.tableName)

def read\_item(self, itemId):

return baseData.select\_one(self.connection, self.tableName, itemId)

def update\_item(self, itemId, name, price, quantity, color, material, description):

try:

assert price > 0, '\033[91m price must be more zero \033[0m'

assert quantity > 0, '\033[91m quantity must be > 0 \033[0m'

assert Additional.findExistingId(self.connection, self.tableName, itemId), \

'\033[91m id isn\'t exist \033[0m'

baseData.update\_one\_item(

self.connection, itemId, name, price, quantity, color, material, description)

return True

except Exception as err:

print(err)

return False

def delete\_item(self, itemId):

try:

assert not Additional.findExistingIdOrderTable(self.connection, self.tableName, itemId), \

'\033[91m item id bind with order \033[0m'

assert Additional.findExistingId(self.connection, self.tableName, itemId), \

'\033[91m id isn\'t exist \033[0m'

baseData.delete\_one(self.connection, itemId, self.tableName)

return True

except Exception as err:

print(err)

return False

class ModelOrder(Model):

def init(self):

super().init()

self.\_tableName = "Order"

@property

def tableName(self):

return self.\_tableName

def create\_order(self, itemId, shopId, customerId, date):

try:

assert Additional.findExistingId(self.connection, "Item", itemId), '\033[91m item id isn\'t exist \033[0m'

assert Additional.findExistingId(self.connection, "Shop", shopId), '\033[91m shop id isn\'t exist \033[0m'

assert Additional.findExistingId(self.connection, "Customer", customerId), \

'\033[91m customer id isn\'t exist \033[0m'

baseData.insert\_one\_order(self.connection, itemId, shopId, customerId, date)

return True

except Exception as err:

print(err)

return False

def create\_many\_orders(self, count):

try:

itemId = Additional.findExistRow(self.connection, "Item")

shopId = Additional.findExistRow(self.connection, "Shop")

customerId = Additional.findExistRow(self.connection, "Customer")

assert Additional.findExistingId(self.connection, "Item", itemId), '\033[91m item id isn\'t exist \033[0m'

assert Additional.findExistingId(self.connection, "Shop", shopId), '\033[91m shop id isn\'t exist \033[0m'

assert Additional.findExistingId(self.connection, "Customer", customerId), \

'\033[91m customer id isn\'t exist \033[0m'

baseData.insert\_many\_random\_orders(self.connection, count, itemId, customerId, shopId)

return True

except Exception as err:

print(err)

return False

def read\_orders(self):

return baseData.select\_all(self.connection, self.tableName)

def read\_order(self, orderId):

return baseData.select\_one(self.connection, self.tableName, orderId)

def read\_order\_another\_table(self, first, second=None):

return Additional.findRowBetweenNumbers(self.connection, first, second)

def read\_order\_itemName(self, name):

return Additional.findItemName(self.connection, name)

def update\_order(self, orderId, itemId, shopId, customerId, date):

try:

assert Additional.findExistingId(self.connection, self.tableName, orderId), \

'\033[91m id isn\'t exist \033[0m'

assert Additional.findExistingId(self.connection, "Item", itemId), '\033[91m item id isn\'t exist \033[0m'

assert Additional.findExistingId(self.connection, "Shop", shopId), '\033[91m shop id isn\'t exist \033[0m'

assert Additional.findExistingId(self.connection, "Customer", customerId), \

'\033[91m customer id isn\'t exist \033[0m'

baseData.update\_one\_order(

self.connection, orderId, itemId, shopId, customerId, date)

return True

except Exception as err:

print(err)

return False

def delete\_order(self, orderId):

try:

assert Additional.findExistingId(self.connection, self.tableName, orderId), \

'\033[91m id isn\'t exist \033[0m'

baseData.delete\_one(self.connection, orderId, self.tableName)

return True

except Exception as err:

print(err)

return False

Файл **view.py**

class View(object):

@staticmethod

def show\_list\_shops(items):

if items:

print("----------------SHOPS------------------")

print("|ShopId\t|", "ShopName |", "ShopStreet |")

for row in items:

print("|", row.ShopId, "\t|", row.ShopName, " " \* (11 - len(row.ShopName)), "|", row.ShopStreet, " " \* (11 - len(row.ShopStreet)), "|")

print("---------------------------------------")

else:

print("There is nothing with this id")

@staticmethod

def display\_shop\_updated(shopId, newName, newStreet):

print('--- --- --- --- --- --- --- --- --- --- --')

print('Change {} name: {}'

.format(shopId, newName))

print('Change {} street: {}'

.format(shopId, newStreet))

print('--- --- --- --- --- --- --- --- --- --- --')

@staticmethod

def show\_list\_customers(items):

if items:

print("----------------CUSTOMERS----------------------------------------")

print("|CustomerId\t|", "CustomerName\t|", "CustomerPhone |", "CustomerSex |")

for row in items:

print("|", row.CustomerId, "\t\t|", row.CustomerName, "\t|", row.CustomerPhone, " " \* (17 - len(row.CustomerPhone)), "|", row.CustomerSex,

" " \* (10 - len(row.CustomerSex)), "\t|")

print("-----------------------------------------------------------------")

else:

print("There is nothing with this id")

@staticmethod

def display\_customer\_updated(shopId, newName, newPhone, newSex):

print('--- --- --- --- --- --- --- --- --- --- --')

print('Change {} name: {}'

.format(shopId, newName))

print('Change {} phone: {}'

.format(shopId, newPhone))

print('Change {} sex: {}'

.format(shopId, newSex))

print('--- --- --- --- --- --- --- --- --- --- --')

@staticmethod

def display\_item\_updated(itemId, old, name, price, quantity, color, material, description):

print('--- --- --- --- --- --- --- --- --- --- --')

print('Change {} name: {} --> {}'

.format(itemId, old[0].ItemName, name))

print('Change {} price: {} --> {}'

.format(itemId, old[0].ItemPrice, price))

print('Change {} quantity: {} --> {}'

.format(itemId, old[0].ItemQuantity, quantity))

print('Change {} color: {} --> {}'

.format(itemId, old[0].ItemColor, color))

print('Change {} material: {} --> {}'

.format(itemId, old[0].ItemMaterial, material))

print('Change {} description: {} --> {}'

.format(itemId, old[0].ItemDescriptions, description))

print('--- --- --- --- --- --- --- --- --- --- --')

@staticmethod

def show\_list\_items(items):

if items:

print("------------------------------------ITEMS-----------------------------------------")

print("|ItemId\t|", "ItemName |", "ItemPrice |", "ItemQuantity |", "ItemColor |",

"ItemMaterial |",

"ItemDescriptions |")

for row in items:

print("|", row.ItemId, "\t|", row.ItemName, " " \* (10 - len(row.ItemName)), "|", row.ItemPrice, "\t|", row.ItemQuantity,

"\t\t\t|", row.ItemColor, " " \* (10 - len(row.ItemColor)), "|", row.ItemMaterial, " " \* (12 - len(row.ItemMaterial)),

"|", row.ItemDescriptions, "|")

print("----------------------------------------------------------------------------------")

else:

print("There is nothing with this id")

@staticmethod

def show\_list\_orders(orders):

if orders:

print("--------------------ORDERS------------------------")

print("|OrderId|", "CustomerId|", "ItemId|", "ShopId|", "OrderDate |")

for row in orders:

print("|", row.orderid, "\t|", row.CustomerId, "\t\t|", row.ItemId, "\t|", row.ShopId,

"\t|", row.orderdate, "|")

print("--------------------------------------------------")

else:

print("There is nothing with this id")

@staticmethod

def display\_order\_updated(orderId, itemId, shopId, customerId, date):

print('--- --- --- --- --- --- --- --- --- --- --')

print('Change {} item: {}'

.format(orderId, itemId))

print('Change {} shop: {}'

.format(orderId, shopId))

print('Change {} customer: {}'

.format(orderId, customerId))

print('Change {} date: {}'

.format(orderId, date))

print('--- --- --- --- --- --- --- --- --- --- --')

@staticmethod

def delete\_connection():

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print("PostgresSQL connection is closed")

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

@staticmethod

def display\_stored(name, tableName):

print('\033[92m++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')

print('Hooray! You create new {} - {} to our {} list!'

.format(tableName.lower(), name, tableName))

print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\033[0m')

@staticmethod

def display\_many\_stored(count, tableName):

print('\033[92m++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')

print('Hooray! You create {} new {} in {} table!'

.format(tableName.lower(), count, tableName))

print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\033[0m')

@staticmethod

def display\_deletion(anyId, tableName):

print('\033[92m--------------------------------------------------------------')

print('We have just removed {} with id:{} from table {}'

.format(tableName.lower(), anyId, tableName))

print(']--------------------------------------------------------------\033[0m')

@staticmethod

def show\_list\_items\_orders(items, lastName):

if items:

print("--------------------ORDERS+ITEMS---------------------------------")

print("|OrderId|", "CustomerId|", "ShopId|", "OrderDate |", "ItemId |", lastName)

for row in items:

print("|", row[0], "\t|", row[1], "\t\t|", row[2],

"\t|", row[3], "|", row[4], "\t\t|", row[5],

"\t\t\t|")

print("-----------------------------------------------------------------")

else:

print("There is nothing with this id")