

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**»**

Виконав студент ІII курсу

ФПМ групи КВ-83

Гончаров В.

Перевірив: Павловський В.І.

Київ – 2020

**Створення додатку бази даних, орієнтованого на взаємодію з 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 2-х 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:
2. контроль при введенні - валідація даних;
3. 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.

З боку батьківської таблиці необхідно контролювати **вилучення** (**ON DELETE**) рядків за умови наявності даних у підлеглій таблиці.

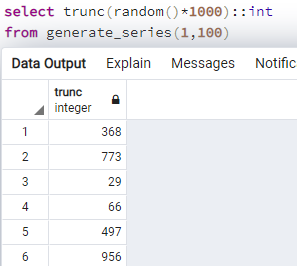
З боку підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** до неї нових даних.

Унеможливити виведення програмою на екрані системних помилок PostgreSQL шляхом їх перехоплення і адекватної обробки.

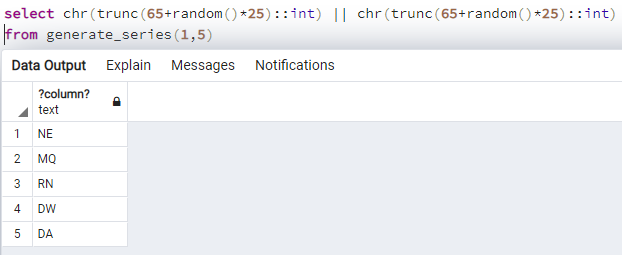
Внесення даних виконується користувачем у консольному вікні програми.

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

Приклад генерації 100 псевдовипадкових чисел:



Приклад генерації 5-ти 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.

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

1. Для реалізації багатокритеріального 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. Після виведення даних 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/). Модель, подання (представлення) 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 лагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.6-3.7

Середовище розробки програмного забезпечення – PyCharm Community Edition 2020.

*Вимоги до оформлення звіту лабораторної роботи у електронному вигляді*

Опис (файл README.md) лабораторної роботи у **репозиторії** **GitHub** включає: назву лабораторної роботи, структуру бази даних з лабораторної роботи №1.

Репозиторій має містить файл звіту у форматі PDF та програмний код файлів мовою Python (або іншою).

Звіт у форматі PDF має містити: титульний аркуш, завдання та відповіді на вимоги до звітування щодо пунктів 1-4 деталізованого завдання:

*Вимоги до пункту №1 деталізованого завдання:*

* ілюстрації обробки виняткових ситуацій (помилок) при уведенні/вилучення даних;
* ілюстрації валідації даних при уведенні користувачем.

*Вимоги до пункту №2 деталізованого завдання:*

* копії екрану (ілюстрації) з фрагментами згенерованих даних таблиць.

*Вимоги до пункту №3 деталізованого завдання:*

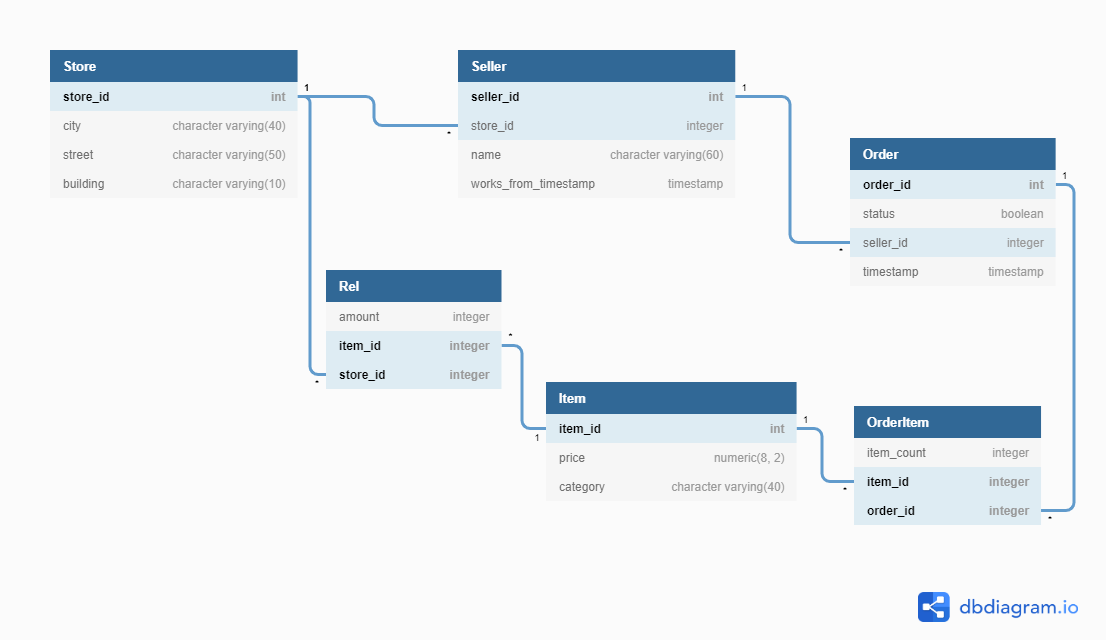
* ілюстрації уведення пошукового запиту та результатів виконання запитів.

*Вимоги до пункту №4 деталізованого завдання:*

* ілюстрації програмного коду з репозиторію Git.

# Нормалізована модель даних БД «Магазин електроніки»

Програма використовує базу даних ‘Магазин електроніки’, розроблену на попередній лабораторній роботі. На рисунку зображена нормалізована модель бази даних.



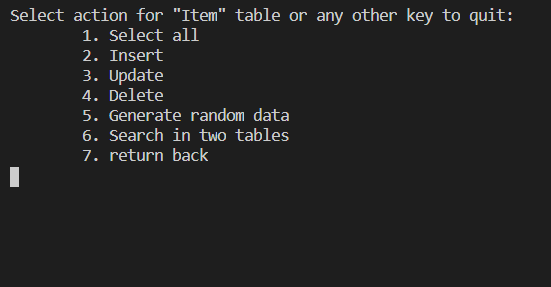
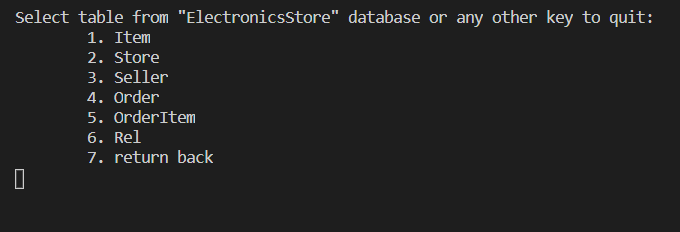
Опис програми

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL та реалізовує функціональні вимоги, що наведені у завданні. Програма складається з чотирьох файлів:

1. Controller.py – файл, який містить контролер для таблиць.
2. Main.py – точка входу в програму.
3. Model.py – модель даних, а також методи доступу до них (sql – запити).
4. View.py – вид – файл, який відповідає за консольний інтерфейс.

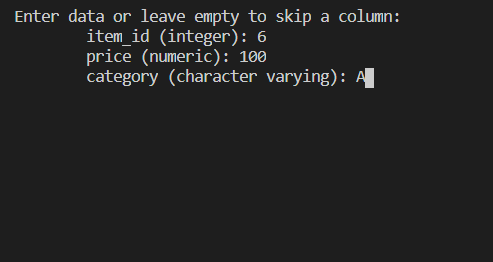
# Результати роботи програми

Запуск програми:

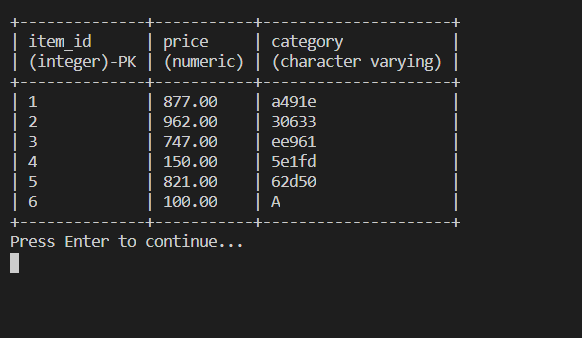
****

# Завдання 1

## Додавання даних до БД:

****

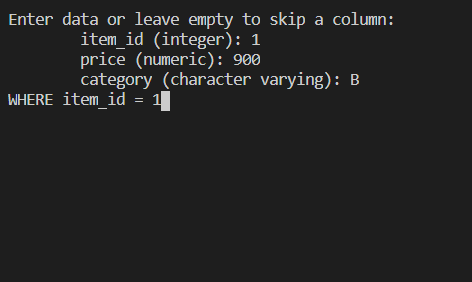
Результат:

****

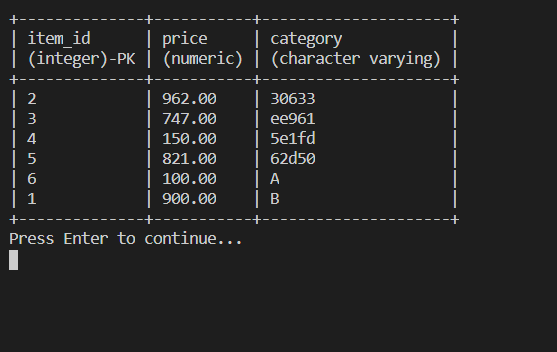
**SQL-запит**

Insert into item(item\_id, price, category) values (6,100,’A’);

## Редагування даних:

****

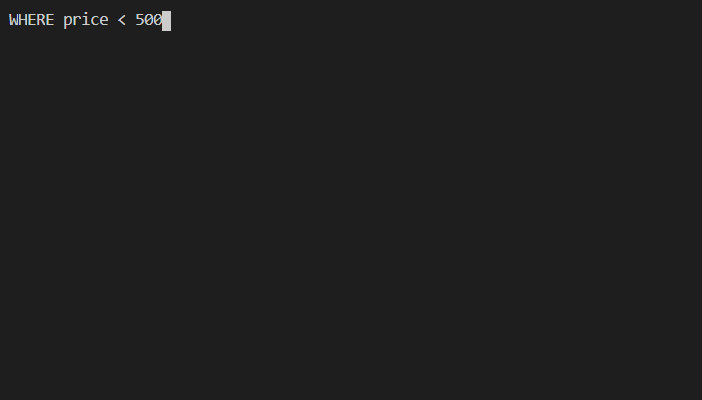
Результат

****

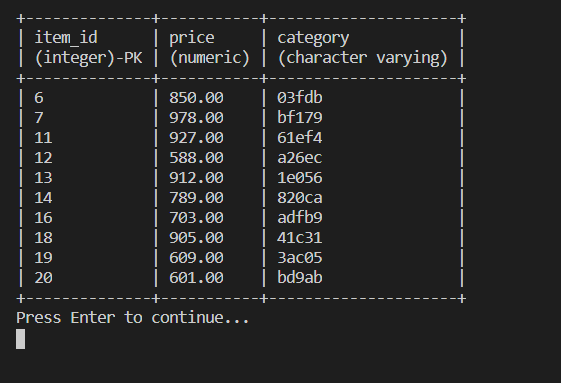
SQL-запит:

Update item set price = 600 and category = B where item\_id = 1

## Видалення даних:

****

Результат:

****

SQL-запит:

Delete from item where price < 500;

## Контроль вилучення (ON DELETE) рядків батьківської таблиці за умови наявності даних у підлеглій таблиці.

Як видно, база даних передбачає видалення зв’язаних таблиць при видаленні ‘батьківської’.

CREATE TABLE public."Seller"

(

seller\_id SERIAL,

store\_id integer,

name character varying(60) COLLATE pg\_catalog."default",

works\_from\_timestamp timestamp without time zone,

CONSTRAINT "Seller\_pkey" PRIMARY KEY (seller\_id),

CONSTRAINT "FK\_StoreSeller" FOREIGN KEY (store\_id)

REFERENCES public."Store" (store\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE CASCADE

NOT VALID

)

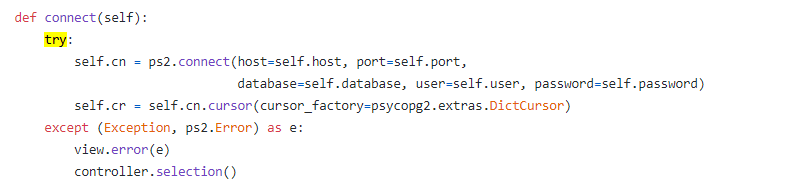
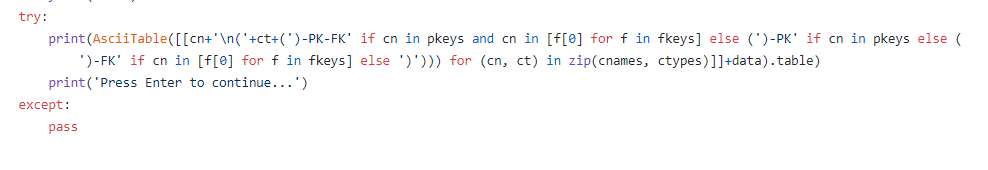
WITH (

OIDS = FALSE

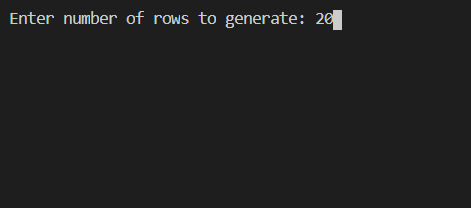
)

## Ілюстрації обробки виняткових ситуацій (помилок) при уведенні/вилучення даних:

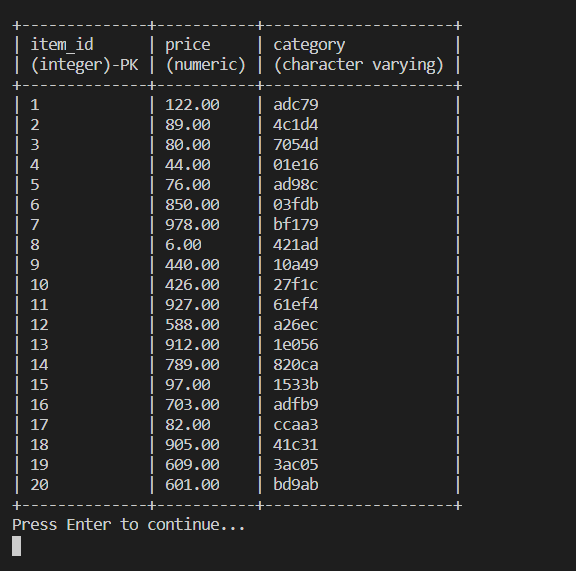
Ілюстрація обробки винятковиї ситуацій(помилок) при введенні/видалянні, генерації даних в коді.



Ілюстрація обробки виняткових ситуацій при зчитуванні даних з таблиці.

Завдання 2. Пакетне генерування даних: ****

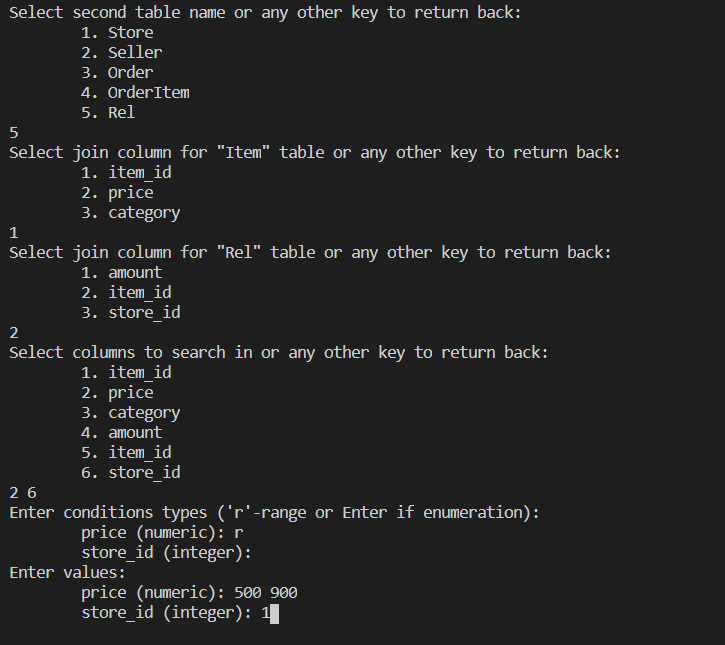
**Результат**

****

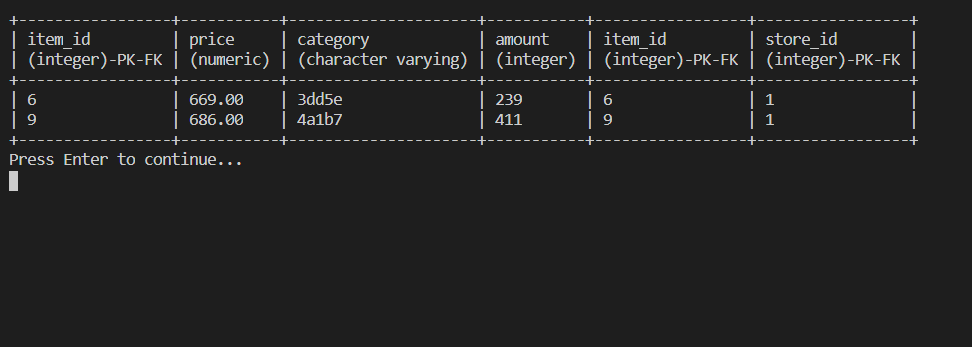
# SQL-запит:

Insert into items (select (random()\*1000)::int , SUBSTRING(md5(random()::text),1,5) from generate\_series(1, 20))

# **Завдання 3. Пошук за двома-трьома атрибутами одночасно**

****

**Результат**

****

SQL-запит:

Select \* from item

Join rel on rel.item\_id = item.item\_id

Where price > 500 and price < 900 and store\_id = 1

# Завдання 4. Ілюстрація програмного коду з репозиторію GitHub: <https://github.com/2v8h5s/database>

## Controller.py

import model

import view

def selection(first=None, \*args):

try:

if first:

first(\*args)

case = input()

return case

except:

return -1

def enter\_col\_values(cnames, ctypes, first=None, empty\_check=True):

try:

if first:

first()

res = []

for cn, ct in zip(cnames, ctypes):

sel\_tmp = selection(view.col\_input, cn, ct)

if not empty\_check or (empty\_check and sel\_tmp != ''):

res.append((cn, sel\_tmp))

return res

except:

return []

def menu\_level1():

case1 = selection(view.section\_level1)

if case1 == '1':

host = selection(view.s\_host)

port = selection(view.s\_port)

database = selection(view.s\_database)

user = selection(view.s\_user)

password = selection(view.s\_password)

elif case1 == '2':

host, port, database, user, password = 'localhost', '5432', 'ElectronicsStore', 'postgres', '10'

else:

view.end()

return

db = model.sql(host, port, database, user, password)

menu\_level2(db)

def menu\_level2(db):

try:

tables\_lst = db.get\_table\_names()

case2 = selection(view.section\_level2, db.database, tables\_lst)

if case2 == str(len(tables\_lst)+1):

menu\_level1()

elif case2 in map(str, range(1, len(tables\_lst)+1)):

menu\_level3(db, tables\_lst[int(case2)-1])

else:

view.end()

return

except:

menu\_level1()

def menu\_level3(db, table):

try:

case3 = selection(view.section\_level3, table)

if case3 == '1':

db.select\_all(table)

selection()

elif case3 == '2':

col\_names, col\_types = db.get\_column\_names(table), db.get\_column\_types(table)

db.insert(table, enter\_col\_values(col\_names, col\_types, view.header\_section\_data))

elif case3 == '3':

col\_names, col\_types = db.get\_column\_names(table), db.get\_column\_types(table)

db.update(table, enter\_col\_values(col\_names, col\_types, view.header\_section\_data),

selection(view.section\_where, False))

elif case3 == '4':

db.delete(table, selection(view.section\_where, True))

elif case3 == '5':

db.gen\_random(table, selection(view.section\_gen\_len))

elif case3 == '6':

try:

col\_names, col\_types = db.get\_column\_names(table), db.get\_column\_types(table)

sec\_cols = [k for k in db.get\_table\_names() if k != table]

table2 = sec\_cols[int(selection(view.section\_get\_table2, sec\_cols))-1]

col\_names2, col\_types2 = db.get\_column\_names(table2), db.get\_column\_types(table2)

rel1 = col\_names[int(selection(view.section\_join, table, col\_names))-1]

rel2 = col\_names2[int(selection(view.section\_join, table2, col\_names2))-1]

coln\_conc, colt\_conc = col\_names+col\_names2, col\_types+col\_types2

search\_coln, search\_colt = list(zip(\*[(coln\_conc[int(x)-1], colt\_conc[int(x)-1])\

for x in selection(view.section\_columns, coln\_conc).split()]))

db.search\_in\_two\_tables(table, table2, rel1, rel2, search\_coln, enter\_col\_values(

search\_coln, search\_colt, view.header\_section\_search\_two, empty\_check=False),

enter\_col\_values(search\_coln, search\_colt, view.enter\_values, empty\_check=False))

selection()

except:

pass

elif case3 == '7':

menu\_level2(db)

return

else:

view.end()

return

menu\_level3(db, table)

except:

menu\_level2(db)

## Main.py

import controller

if \_\_name\_\_ == '\_\_main\_\_':

controller.menu\_level1()

## Model.py

import psycopg2 as ps2

import psycopg2.extras

import psycopg2.sql

import view

import controller

class sql:

def \_\_init\_\_(self, host, port, database, user, password):

self.host, self.port, self.database, self.user, self.password = host, port, database, user, password

def connect(self):

try:

self.cn = ps2.connect(host=self.host, port=self.port,

database=self.database, user=self.user, password=self.password)

self.cr = self.cn.cursor(cursor\_factory=psycopg2.extras.DictCursor)

except (Exception, ps2.Error) as e:

view.error(e)

controller.selection()

def close(self):

try:

self.cr.close()

self.cn.close()

except (Exception, ps2.Error) as e:

pass

def get\_table\_names(self):

try:

self.connect()

self.cr.execute(psycopg2.sql.SQL(

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

res = [v[0] for v in self.cr.fetchall()]

self.close()

return res

except (Exception, ps2.Error) as e:

pass

def get\_column\_names(self, table):

try:

self.connect()

self.cr.execute(psycopg2.sql.SQL("SELECT column\_name FROM information\_schema.columns WHERE \

(table\_schema='public' and table\_name= %s) ORDER BY ordinal\_position;"), (table,))

res = [v[0] for v in self.cr.fetchall()]

self.close()

return res

except (Exception, ps2.Error) as e:

pass

def get\_column\_types(self, table):

try:

self.connect()

self.cr.execute(psycopg2.sql.SQL("SELECT data\_type FROM information\_schema.columns WHERE \

(table\_schema='public' and table\_name= %s) ORDER BY ordinal\_position;"), (table,))

res = [v[0] for v in self.cr.fetchall()]

self.close()

return res

except (Exception, ps2.Error) as e:

pass

def get\_pkeys(self, table):

try:

self.connect()

self.cr.execute(psycopg2.sql.SQL("SELECT kcu.column\_name\

from information\_schema.table\_constraints tco\

join information\_schema.key\_column\_usage kcu on kcu.constraint\_name=tco.constraint\_name\

and kcu.constraint\_schema=tco.constraint\_schema and kcu.constraint\_name=tco.constraint\_name\

where tco.constraint\_type='PRIMARY KEY' and kcu.table\_name='{}'").format(psycopg2.sql.SQL(table)))

res = [v[0] for v in self.cr.fetchall()]

self.close()

return res

except (Exception, ps2.Error) as e:

pass

def get\_fkeys(self, table):

try:

self.connect()

self.cr.execute(psycopg2.sql.SQL("SELECT stb.column\_name, ftb.table\_name, ftb.column\_name\

from information\_schema.key\_column\_usage ftb join\

(SELECT tco.column\_name, tco.constraint\_name, rct.unique\_constraint\_name\

from information\_schema.key\_column\_usage tco join information\_schema.referential\_constraints\

rct on tco.constraint\_name=rct.constraint\_name where tco.table\_name='{}') stb\

on ftb.constraint\_name=stb.unique\_constraint\_name").format(psycopg2.sql.SQL(table)))

res = self.cr.fetchall()

self.close()

return res

except (Exception, ps2.Error) as e:

pass

def get\_col\_info(self, table):

return self.get\_column\_names(table), self.get\_column\_types(table), \

self.get\_pkeys(table), self.get\_fkeys(table),

def select\_all(self, table):

col\_names, col\_types, pkeys, fkeys = self.get\_col\_info(table)

try:

self.connect()

self.cr.execute(

psycopg2.sql.SQL("SELECT \* FROM public.{};")

.format(psycopg2.sql.Identifier(table)))

data = self.cr.fetchall()

view.print\_table(col\_names, col\_types, pkeys, fkeys, data)

except (Exception, ps2.Error) as e:

view.error(e)

controller.selection()

finally:

self.close()

def insert(self, table, sels):

try:

self.connect()

self.cr.execute(psycopg2.sql.SQL("INSERT INTO public.{} ({}) VALUES ({});")

.format(psycopg2.sql.Identifier(table),

psycopg2.sql.SQL(

", ".join([s[0] for s in sels])),

psycopg2.sql.SQL(", ".join(["%s" for i in range(len(sels))]))),

tuple([s[1] for s in sels]))

self.cn.commit()

except (Exception, ps2.Error) as e:

view.error(e)

controller.selection()

finally:

self.close()

def update(self, table, sels, where):

try:

self.connect()

self.cr.execute(psycopg2.sql.SQL("UPDATE public.{} SET {} {};")

.format(psycopg2.sql.Identifier(table),

psycopg2.sql.SQL(

", ".join([s[0]+"=%s" for s in sels])),

(psycopg2.sql.SQL("WHERE "+where) if where else psycopg2.sql.SQL(""))),

tuple([s[1] for s in sels]))

self.cn.commit()

except (Exception, ps2.Error) as e:

view.error(e)

controller.selection()

finally:

self.close()

def delete(self, table, where):

try:

self.connect()

self.cr.execute(psycopg2.sql.SQL("DELETE FROM {} {};")

.format(psycopg2.sql.Identifier(table),

(psycopg2.sql.SQL("WHERE "+where) if where else psycopg2.sql.SQL(""))))

self.cn.commit()

except (Exception, ps2.Error) as e:

view.error(e)

controller.selection()

finally:

self.close()

def gen\_random(self, table, gen\_len):

col\_names, col\_types, pkeys, fkeys = self.get\_col\_info(table)

self.delete(table, '')

try:

self.connect()

self.cr.execute(psycopg2.sql.SQL(

"INSERT INTO public.{} (SELECT {} FROM generate\_series(1, {}));")

.format(psycopg2.sql.Identifier(table),

psycopg2.sql.SQL(", ".join([(psycopg2.sql.SQL("(SELECT {} FROM public.{} ORDER BY RANDOM()+generate\_series LIMIT 1)")

.format(psycopg2.sql.SQL([f[2] for f in fkeys if f[0] == c][0]),

psycopg2.sql.Identifier([f[1] for f in fkeys if f[0] == c][0])).as\_string(self.cn)

if c in [f[0] for f in fkeys] else (("generate\_series" if t in ['integer', 'numeric'] else "generate\_series::text")

if c in pkeys else (("NOW()+(random()\*(interval '90 days'))+'30 days'" if 'timestamp' in t

else ("cast(random()::int as boolean)" if t == 'boolean'

else ("SUBSTRING(md5(random()::text),1,5)" if t in ['text', 'character varying']

else "(random()\*1000)::int"))))))

for c, t in zip(col\_names, col\_types)])), psycopg2.sql.SQL(gen\_len)))

self.cn.commit()

except (Exception, ps2.Error) as e:

view.error(e)

controller.selection()

finally:

self.close()

def search\_in\_two\_tables(self, table, table2, rel1, rel2, search\_col, mods, values):

col\_names, col\_types, pkeys, fkeys = self.get\_col\_info(table)

col\_names2, col\_types2, pkeys2, fkeys2 = self.get\_col\_info(table2)

try:

self.connect()

self.cr.execute(

psycopg2.sql.SQL(

"SELECT \* FROM public.{0} JOIN public.{1} ON public.{0}.{2} = public.{1}.{3} WHERE {4};")

.format(psycopg2.sql.Identifier(table), psycopg2.sql.Identifier(table2), psycopg2.sql.SQL(rel1), psycopg2.sql.SQL(rel2),

psycopg2.sql.SQL(" AND ".join([(c+"=%s" if m[1] == '' else "("+c+" BETWEEN "+v[1].split()[0]+" AND "+v[1].split()[1]+")")

for c, m, v in zip(search\_col, mods, values)]))),

tuple([v[1] for m, v in zip(mods, values) if m[1] != 'r']))

data = self.cr.fetchall()

view.print\_table(col\_names+col\_names2, col\_types+col\_types2, pkeys+pkeys2, fkeys+fkeys2, data)

except (Exception, ps2.Error) as e:

view.error(e)

finally:

self.close()

## View.py

import os

from terminaltables import AsciiTable

def section\_level1(\*args):

os.system('cls')

print('Select action or any other key to quit:')

print('\t1. Define database details manually')

print('\t2. Use defaults')

def section\_level2(\*args):

os.system('cls')

print('Select table from "%s" database or any other key to quit: ' % (args[0]))

for ind, row in enumerate(args[1]):

print('\t'+str(ind+1)+'. '+row)

print('\t'+str(len(args[1])+1)+'. return back')

def section\_level3(\*args):

os.system('cls')

print('Select action for "%s" table or any other key to quit:' % (args[0]))

print('\t1. Select all')

print('\t2. Insert')

print('\t3. Update')

print('\t4. Delete')

print('\t5. Generate random data')

print('\t6. Search in two tables')

print('\t7. return back')

def section\_input(\*args):

print('\t'+args[0]+': ', end='')

def col\_input(\*args):

print('\t'+args[0]+' ('+args[1]+'): ', end='')

def section\_where(\*args):

if args[0]:

os.system('cls')

print('WHERE ', end='')

def section\_gen\_len(\*args):

os.system('cls')

print('Enter number of rows to generate: ', end='')

def section\_get\_table2(\*args):

os.system('cls')

print('Select second table name or any other key to return back: ')

for ind, row in enumerate(args[0]):

print('\t'+str(ind+1)+'. '+''.join(row))

def section\_join(\*args):

print('Select join column for "%s" table or any other key to return back: '%args[0])

for ind, ct in enumerate(args[1]):

print('\t'+str(ind+1)+'. '+ct)

def section\_columns(\*args):

print('Select columns to search in or any other key to return back: ')

for ind, row in enumerate(args[0]):

print('\t'+str(ind+1)+'. '+''.join(row))

def section\_text\_search\_col(\*args):

print('Select column to search in or any other key to return back: ')

for ind, row in enumerate(args[0]):

print('\t'+str(ind+1)+'. '+''.join(row))

def section\_search\_mode(\*args):

os.system('cls')

print('Enter search mode (\'1\'-not word match, \'2\'-word match): ', end='')

def enter\_text(\*args):

print('Enter words to search: ', end='')

def header\_section\_data():

os.system('cls')

print('Enter data or leave empty to skip a column:')

def header\_section\_search\_two():

print('Enter conditions types (\'r\'-range or Enter if enumeration):')

def enter\_values():

print('Enter values: ')

def s\_host(\*args):

os.system('cls')

print('Host: ', end='')

def s\_port(\*args):

os.system('cls')

print('Port: ', end='')

def s\_database(\*args):

os.system('cls')

print('DataBase: ', end='')

def s\_user(\*args):

os.system('cls')

print('User: ', end='')

def s\_password(\*args):

os.system('cls')

print('Password: ', end='')

def error(e):

print('[ERROR]', str(e))

def print\_table(cnames, ctypes, pkeys, fkeys, data):

os.system('cls')

try:

print(AsciiTable([[cn+'\n('+ct+(')-PK-FK' if cn in pkeys and cn in [f[0] for f in fkeys] else (')-PK' if cn in pkeys else (

')-FK' if cn in [f[0] for f in fkeys] else ')'))) for (cn, ct) in zip(cnames, ctypes)]]+data).table)

print('Press Enter to continue...')

except:

pass

def end():

print('Closing')