Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Базы данных

Отчет по лабораторной работе №3
Генерация тестовых данных

Работу выполнил:
Графов Д.И.
Группа: 33531/2
Преподаватель:
Мяснов А.В.

Санкт-Петербург
2019

Содержание

1. Цель работы

Сформировать набор данных, позволяющий производить операции на реальных объемах данных.

2. Программа работы

1. Реализация в виде программы параметризуемого генератора, который позволит сформировать набор связанных данных в каждой таблице.
2. Частные требования к генератору, набору данных и результирующему набору данных:
   - количество записей в справочных таблицах должно соответствовать ограничениям предметной области
   - количество записей в таблицах, хранящих информацию об объектах или субъектах должно быть параметром генерации
   - значения для внешних ключей необходимо брать из связанных таблиц

3. Выполнение работы

В качестве языка программирования для параметризуемого создания генератора был выбран Python 3.6 и библиотека psycopg2 - самая популярная библиотека для работы с PostgreSQL.

В ходе выполнения работы была написана программа, реализующая генератор. Код программы приведён ниже.

Листинг 1. fill.py

```
import random
import string
from datetime import datetime

import psycopg2
from psycopg2 import sql


def readConfig():
    result = {}
    file = open('../../user.ini', 'r')
    for line in file.readlines():
        splitted = line.split('=')
        result[splitted[0]] = splitted[1].replace('\n', '')
    return result


def rand_string():
    return ''.join(
        random.choice(string.ascii_letters + string.digits) for _ in
```

```
        range(random.randint(1, MAX_LEN_OF_RAND_STR)))


def random_date():
    return datetime(random.randint(2005, 2025), random.randint(1, 12), random.randint(1, 28), random.ra
                    random.randint(0, 59), random.randint(0, 59))


def fill_components_and_drinks(components_size=100, drinks_size=100, max_components_per_drink=10):
    with conn.cursor() as cursor:
        conn.autocommit = True
        for i in range(components_size):
            rand_component = rand_string()
            rand_alco = random.randint(0, 100)
            components.append((rand_component, rand_alco))

        insert_components = sql.SQL('INSERT INTO components(title, alcohol) VALUES {}').format(
            sql.SQL(',').join(map(sql.Literal, components))
        )
        cursor.execute(insert_components)

        for i in range(drinks_size):
            rand_drink = rand_string()
            rand_vol = random.randint(100, 1000)
            rand_alco = random.randint(0, 100)
            drinks.append((rand_drink, rand_vol, rand_alco, drink_type[random.randint(0, len(drink_type

        insert_drinks = sql.SQL('INSERT INTO drinks(title, volume, alcohol, drink_type) VALUES {}').for
            sql.SQL(',').join(map(sql.Literal, drinks))
        )
        cursor.execute(insert_drinks)

        components_drinks = []  # [component_id, drink_id, quantity]
        for drink in drinks:
            cursor.execute("select drink_id from drinks where title = '{}'".format(drink[0]))
            rand_drink_id = cursor.fetchone()
            for i in range(0, max_components_per_drink):
                cursor.execute("select component_id from components where title = '{}';".format(
                    components[random.randint(0, len(components) - 1)][0]))
                rand_component_id = cursor.fetchone()
                quantity = random.randint(0, 10)
                components_drinks.append((rand_component_id, rand_drink_id, quantity))

        insert_components_drinks = sql.SQL(
            'INSERT INTO components_drinks(component_id, drink_id, quantity) VALUES {}').format(
            sql.SQL(',').join(map(sql.Literal, components_drinks))
        )
        cursor.execute(insert_components_drinks)
        print("Successfully filled components and drinks!")

```

```
71
72  def fill_places(places_size=100):
73      with conn.cursor() as cursor:
74          conn.autocommit = True
75          for i in range(places_size):
76              rand_title = rand_string()
77              rand_address = rand_string()
78              places.append((rand_title, rand_address))
79
80          insert_places = sql.SQL('INSERT INTO places(title, address) VALUES {}').format(
81              sql.SQL(',').join(map(sql.Literal, places))
82          )
83          cursor.execute(insert_places)
84          print("Successfully filled places!")
85
86
87  def fill_food(food_size=100):
88      with conn.cursor() as cursor:
89          conn.autocommit = True
90          for i in range(food_size):
91              rand_title = rand_string()
92              rand_price = random.randint(1, 10000)
93              food.append((rand_title, rand_price))
94
95          insert_food = sql.SQL('INSERT INTO food(title, average_price) VALUES {}').format(
96              sql.SQL(',').join(map(sql.Literal, food))
97          )
98          cursor.execute(insert_food)
99          print("Successfully filled food!")
100
101
102 def fill_places_food(size=100, max_food_per_place=10):
103     places_food = []  # [place_id, food_id)
104     records = 0
105     with conn.cursor() as cursor:
106         conn.autocommit = True
107         for k in range(0, len(places) - 1):
108             cursor.execute(
109                 "select place_id from places where title = '{}';".format(places[random.randint(0, len(p
110             rand_place_id = cursor.fetchone()
111             for i in range(0, max_food_per_place):
112                 cursor.execute(
113                     "select food_id from food where title = '{}'".format(food[random.randint(0, len(foo
114                 rand_food_id = cursor.fetchone()
115                 if records < size:
116                     places_food.append((rand_place_id, rand_food_id))
117                     records += 1
118
119         insert_places_food = sql.SQL('INSERT INTO places_food(place_id, food_id) VALUES {}').format(
120             sql.SQL(',').join(map(sql.Literal, places_food))
```

```
        )
        cursor.execute(insert_places_food)
        print("Successfully filled places_food!")


def fill_places_drinks(size=100, max_drinks_per_place=10):
    places_drinks = []  # [place_id, drink_id)
    records = 0
    with conn.cursor() as cursor:
        conn.autocommit = True
        for k in range(0, len(places) - 1):
            cursor.execute(
                "select place_id from places where title = '{}';".format(places[random.randint(0, len(p
            rand_place_id = cursor.fetchone()
            for i in range(0, max_drinks_per_place):
                cursor.execute(
                    "select drink_id from drinks where title = '{}'".format(
                        drinks[random.randint(0, len(drinks) - 1)][0]))
                rand_drink_id = cursor.fetchone()
                if records < size:
                    places_drinks.append((rand_place_id, rand_drink_id))
                    records += 1

        insert_places_drinks = sql.SQL('INSERT INTO places_drinks(place_id, drink_id) VALUES {}').forma
            sql.SQL(',').join(map(sql.Literal, places_drinks))
        )
        cursor.execute(insert_places_drinks)
        print("Successfully filled places_drinks!")


def fill_discounts(size=10):
    records = 0
    discounts = []
    with conn.cursor() as cursor:
        while records < size:
            cursor.execute(
                "select place_id from places where title = '{}';".format(places[random.randint(0, len(p
            rand_place_id = cursor.fetchone()
            rand_amount = random.random()
            rand_description = rand_string()
            discounts.append(
                (rand_place_id, drink_type[random.randint(0, len(drink_type) - 1)], rand_amount, rand_d
            records += 1

        insert_discounts = sql.SQL(
            'INSERT INTO discounts(place_id, drink_type, amount, description) VALUES {}').format(
            sql.SQL(',').join(map(sql.Literal, discounts))
        )
        cursor.execute(insert_discounts)
        print("Successfully filled discounts!")
```

```
171
172
173 def fill_supplies_drinks(size=10):
174     supplies_drinks = []
175     with conn.cursor() as cursor:
176         for i in range(0, size):
177             cursor.execute(
178                 "select place_id from places where title = '{}';".format(places[random.randint(0, len(p
179             rand_place_id = cursor.fetchone()
180             rand_amount = random.randint(0, 100)
181             rand_price = random.randint(0, 1000)
182             cursor.execute(
183                 "select drink_id from drinks where title = '{}'".format(drinks[random.randint(0, len(dr
184             rand_drink_id = cursor.fetchone()
185             date = random_date()
186             supplies_drinks.append((rand_place_id, rand_drink_id, rand_amount, rand_price, date))
187
188         insert_supplies_drinks = sql.SQL(
189             'INSERT INTO supplies_drinks(place_id, drink_id, amount, price_per_item, date) VALUES {}').
190             sql.SQL(',').join(map(sql.Literal, supplies_drinks))
191         )
192         cursor.execute(insert_supplies_drinks)
193         print("Successfully filled supplies_drinks!")
194
195
196 def fill_supplies_food(size=10):
197     supplies_food = []
198     with conn.cursor() as cursor:
199         for i in range(0, size):
200             cursor.execute(
201                 "select place_id from places where title = '{}';".format(places[random.randint(0, len(p
202             rand_place_id = cursor.fetchone()
203             rand_amount = random.randint(0, 100)
204             rand_price = random.randint(0, 1000)
205             cursor.execute(
206                 "select food_id from food where title = '{}'".format(food[random.randint(0, len(food) -
207             rand_food_id = cursor.fetchone()
208             date = random_date()
209             supplies_food.append((rand_place_id, rand_food_id, rand_amount, rand_price, date))
210
211         insert_supplies_drinks = sql.SQL(
212             'INSERT INTO supplies_food(place_id, food_id, amount, price_per_item, date) VALUES {}').for
213             sql.SQL(',').join(map(sql.Literal, supplies_food))
214         )
215         cursor.execute(insert_supplies_drinks)
216         print("Successfully filled supplies_food!")
217
218
219 if __name__ == '__main__':
220     config = readConfig()
```

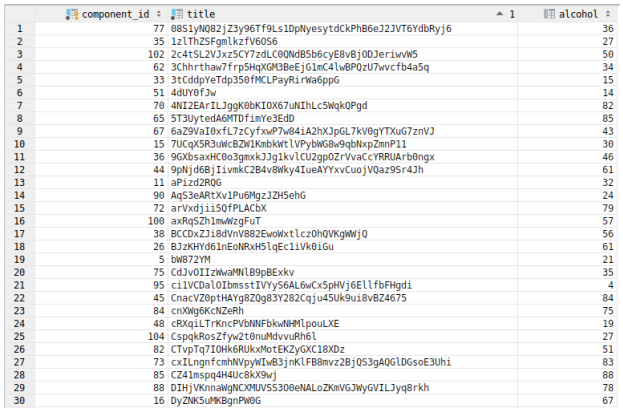
```
    user = config['user']
    password = config['password']
    conn = psycopg2.connect(dbname='wine_card', user=user, password=password, host='localhost')

    # set default settings
    with conn.cursor() as cursor:
        conn.autocommit = True
        cursor.execute(open("../../lab2/src/init.sql", "r").read())
        cursor.execute(open("../../lab2/src/fill.sql", "r").read())
        cursor.execute(open("../../lab2/src/change.sql", "r").read())

        cursor.execute('SELECT unnest(enum_range(NULL::drink_type))::text;')
        drink_type = cursor.fetchall()
        drink_type = [drink_type[i][0] for i in range(len(drink_type))]

    MAX_LEN_OF_RAND_STR = 50

    components = []  # [title, alcohol]
    drinks = []  # [title, volume, alcohol, drink_type]
    food = []  # [title, average_price]
    places = []  # [title, address]

    fill_components_and_drinks()
    fill_places()
    fill_food()
    fill_places_food()
    fill_places_drinks()
    fill_discounts()
    fill_supplies_drinks()
    fill_supplies_food()
```

4. Результат работы

Запустим данную программу через терминал.

```
(base) grafa@KRAB:~/Desktop/wine_card/lab3/src$ python fill.py
Successfully filled components and drinks!
Successfully filled places!
Successfully filled food!
Successfully filled places_food!
Successfully filled places_drinks!
Successfully filled discounts!
Successfully filled supplies_drinks!
Successfully filled supplies_food!
```

Примеры генерируемых данных:



	component_id	title	1	alcohol
1	77	08SiyNQ8ZjZ3y96Tf9Ls1DpNyesytdCkPhB6eJ2JVT6YdbRyj6		36
2	35	1zlThZSFgmlkzfV6OS6		27
3	102	2c4tSL2VJxz5CY7zdLC0QNdB5b6cyE8vBjODJeriwvW5		50
4	62	3Chhrthaw7frp5HqXGM3BeEjG1mC4lwBPQzU7wvcfb4a5q		34
5	33	3tCddpYeTdp350fMCLPayRirWa6ppG		15
6	51	4dUY0fJw		14
7	70	4NI2EArILJggK0bKIOX67uNIhLc5WqkQPgd		82
8	65	5T3UytedA6MTDfimYe3EdD		85
9	67	6aZ9VaI0xfL7zCyfxwP7w04iA2hXJpGL7kV0gYTXuG7znVJ		43
10	15	7UCqX5R3uWcBZW1KmbkWtlVPybWG8w9qbNxpZmnP11		30
11	36	9GXbsaxHC0o3gmxkJJg1kvlCU2gpOZrVvaCcYRRUArb0ngx		46
12	44	9pNjd6BjIivmkC2B4v8Wky4IueAYYxvCuojVQaz9Sr4Jh		61
13	11	aPizd2RQG		32
14	90	AqS3eARtXv1Pu6MgzJZH5ehG		24
15	72	arVxdjii5QfPLACbX		79
16	100	axRqSZh1mwWzgFuT		57
17	38	BCCDxZJi8dVnV882EwoWxtlczOhQVKgWWjQ		56
18	26	BJzKHYd61nEoNRxH5lqEc1iVk0iGu		61
19	5	bW872YM		21
20	75	CdlvOIIzHwaMNlB9pBExkv		35
21	95	ci1VCDalOIDmsstIVYyS6AL6wCx5pHVj6EllfbFHgdi		4
22	45	CnacVZ0ptHAYg8ZQg83Y282Cqju45Uk9ui8vBZ4675		84
23	84	cnXWg6KcNZeRh		75
24	48	cRXqiLTrKncPVbNNFbkwNHMlpouLXE		19
25	104	CspqkRosZfyw2t0nuMdvvuRh6l		27
26	82	CTvpTq7IOHk6RUkxMotEKZyGXC18XDz		51
27	73	cxILngnfcmhNVpyWIwB3jnKlFB8mvz2BjQS3gAQGlDGsoE3Uhi		83
28	85	CZ41mspq4H4Uc8kX9wj		88
29	88	DIHjVKnnaWgNCXMUVSS3O0eNALoZKmVGJWyGVILJyq8rkh		78
30	16	DyZNK5uMKBgnPW0G		67

Рисунок 4.1. Содержимое таблицы components

	supplies_drinks_id	place_id	drink_id	amount	price_per_item 1	date
1	16	99	88	46	50	2021-03-09 21:39:05.000000
2	14	111	87	97	248	2016-03-08 04:19:35.000000
3	12	35	14	89	266	2012-01-28 03:22:00.000000
4	17	46	74	26	314	2011-07-19 20:34:29.000000
5	19	89	64	39	319	2021-04-17 16:07:19.000000
6	18	90	91	56	531	2009-11-18 19:35:33.000000
7	20	56	38	92	581	2013-01-10 07:55:33.000000
8	11	58	62	4	603	2011-08-05 11:32:45.000000
9	13	74	47	30	620	2007-03-27 07:05:25.000000
10	15	71	74	23	755	2012-01-13 20:11:15.000000

Рисунок 4.2. Содержимое таблицы supplies_drinks

	discount_id	place_id	drink_type	amount	description 1
1	18	15	виски	0.7414360619367385	2KeaArmVOq6JhbN0QVa3iR
2	11	67	текила	0.7198427226334619	A2
3	12	70	ром	0.7290235045170657	Esug3keLhJYyEMeZrnr3ZpeDR2QwzjJ0J3o
4	13	71	пиво и сидр	0.182336489667546	HAuo3YxN3lSDmDmnjeWn6Lcy2CtZXidNx8TmI57
5	15	16	бренди и коньяк	0.6130395355830033	hIYkmTHhH1GUYYaCY
6	20	67	шоты	0.12565166052860477	L82r0bf
7	14	62	биттеры	0.1062119366335399	NuRWcIMMGHcAXB7WZwLj3
8	16	54	водка и настойки	0.9232124366668905	vN7PiTctwLUikGry
9	17	18	ром	0.959227174910468	Z8DaatrbaliOMJKdepEoOfkKiMWTeGQahI6Aaggp
10	19	92	ром	0.06219161989266875	zDGbC3oKnf5S8eTDqiJc7TbnQWD2

Рисунок 4.3. Содержимое таблицы discounts

5. Выводы

В ходе выполнения данной работы на языке программирования Python был написан параметризуемый генератор. В качестве параметра данного генератора можно указать количество записей в таблицах, как это и требовало задание.