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:
Еременко Д.Ю.
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. Ход выполнения работы

3.1. Заполняющая программа

```
1  import psycopg2
2  import numpy as np
3  from random_words import RandomWords
4  import sys
5  import json
6
7  # ------------------------- parameters for generation
       ↪ ---------------------------------------
8  MAX_VCH_LEN = 0
9
10 MIN_PRIOR = 0
11 MAX_PRIOR = 0
12
13 MIN_PRICE = 0
14 MAX_PRICE = 0
15
16 MIN_DPRICE = 0
17 MAX_DPRICE = 0
18
19 MIN_DAY_EXP = 0
20 MAX_DAY_EXP = 0
21
22 MIN_AM = 0
23 MAX_AM = 0
24
25 MIN_WEIGHT = 0
26 MAX_WEIGHT = 0
27
28 MIN_ADD_AVAIL = 0
29 MAX_ADD_AVAIL = 0
30
```

```
31 # ---------------------------------- types of fields
   ↪ ----------------------------------------
32 PINT = 0  # param is integer
33 PSTR = 1  # param is word
34 PSEQ = 2  # param is sequence of words
35 PREF = 3  # param is reference
36 PDATE = 4  # param is date
37 PID = 5  # param is id
38
39
40 # ---------------------------------- GETTERS FROM CONSOLE
   ↪ ------------------------------------------------------------
41 def get_input_int(title=None, min=None, max=None):
42 while True:
43 if title != None:
44 print(title)
45 res = input()
46 if res.isdigit():
47 res = int(res)
48
49 if (min != None) and (max != None):
50 if (res >= min and res <= max):
51 return res
52 else:
53 print("%d␣isn't␣between␣[%d,%d]" % (res, min, max))
54 else:
55 return res
56
57 else:
58 print("%s␣not␣an␣integer\n" % res)
59
60
61 def get_input_str(title=None, min_size=1, max_size=15):
62 while True:
63 if title != None:
64 print(title)
65 res = input()
66 if not res.isdigit():
67
68 if (res.__len__() >= min_size and res.__len__() <= max_size):
69 return res
70 else:
71 print("%s␣isn't␣between␣[%d,%d]" % (res, min_size, max_size))
72
73 else:
74 print("%s␣not␣an␣string\n" % res)
75
76
77 # ---------------------------------- CHOOSE FROM CONSOLE
   ↪ ------------------------------------------
78 def choose_variant_from_dict(title, variants):
79 variants_str = ""
80 for n, name in variants.items(): variants_str += "%d:%s\n" % (n, name)
81
82 while True:
83 answ = get_input_int(title="%s\n%s" % (title, variants_str))
84
85 if variants.keys().__contains__(answ):
86 return answ
87 else:
```

```
print("Unexpected value!\n")


def choose_variant_from_turp(title, variants):
"""
Correct processing of illegal value works.
:param title:
:param variants:
:return: one of allowed number
"""
variants_str = ""
for n, name in variants: variants_str += "%d:%s\n" % (n, name)

while True:
answ = get_input_int(title="%s\n%s" % (title, variants_str))

for var in variants:
if var.__contains__(answ):
return answ
else:
print("Unexpected value!\n")


# ---------------------------------- GETTERS
    ↪ ----------------------------------------------------
def get_table_turp(table_name):
cursor.execute("select * from %s;" % table_name)
table = cursor.fetchall()
min_id = 0
max_id = table.__len__()
return (min_id, max_id, table)


def get_free_ids(cursor, table_name):
"""
:param cursor:
:param table_name:
:return: array of free ids in table
"""

cursor.execute("select id from %s;" % table_name)
ids = np.array(cursor.fetchall())
shids = ids + 1
fids = np.empty(0, dtype=int)
for shid in shids:
if not ids.__contains__(shid):
fids = np.append(fids, shid)
return fids


def get_free_id(cursor, table_name):
"""
:param cursor:
:param table_name:
:return: array of free ids in table
"""
cursor.execute("select max(id) from %s" % table_name)
max = cursor.fetchall()[0][0]
if max == None:
return 1
```

```
147 else:
148 return max + 1
149
150
151 def get_random_word(rw, min_size, max_size):
152 word = rw.random_word()
153 while word.__len__() < min_size or word.__len__() > max_size:
154 word = rw.random_word()
155 return word
156
157
158 # ---------------------------------- MAIN METHOD FOR TABLES FILLING
    ↪ ----------------------------------
159 def add_into_table(cursor, rw, table_name, fields, min_av, max_av, bounds=None):
160 """
161 :param cursor: cursor to database
162 :param rw: random word
163 :param table_name: name of fielded table
164 :param fields: dictionary that contains fields in keys and types in values
165 :param bounds: bounds for int fields
166 :param min_av: min amount of randomized lines
167 :param max_av: max amount of randomized lines
168 :return: pass
169 """
170 way = choose_variant_from_dict("CHOOSE WAY OF ADDING FOR TABLE '%s'" %
    ↪ table_name, {1: 'random', 2: 'not random'})
171
172 if way == 1:
173 num = get_input_int(title="HOW MANY?[%d - %d]" % (min_av, max_av), min=min_av,
    ↪ max=max_av)
174
175 for i in range(0, num):
176
177 request = "insert into %s values(" % table_name
178
179 bi = 0
180 for field, partype in fields.items():
181 if partype == PINT:
182 MIN_B = bounds[bi][0]
183 MAX_B = bounds[bi][1]
184 request += "%d," % np.random.randint(MIN_B, MAX_B)
185 bi += 1
186
187 elif partype == PSTR:
188 MIN_B = bounds[bi][0]
189 MAX_B = bounds[bi][1]
190 word = get_random_word(rw, min_size=MIN_B, max_size=MAX_B)
191 request += "\'%s\'," % word
192 bi += 1
193
194 elif partype == PSEQ:
195 MIN_B = bounds[bi][0]
196 MAX_B = bounds[bi][1]
197 seq = ""
198 for i in range(MIN_B):
199 seq += get_random_word(rw, min_size=1, max_size=MAX_VCH_LEN) + " "
200 request += "\'%s\'," % seq[:seq.__len__() - 1]
201 bi += 1
202
203 elif partype == PREF:
```

```
204 (min_id, max_id, lines) = get_table_turp(field)
205 currid = np.random.randint(low=min_id, high=max_id)
206 request += "%d," % lines[currid][0]
207
208 elif partype == PDATE:
209 request += "%s," % "current_date"
210
211 elif partype == PID:
212 rid = get_free_id(cursor, table_name)
213 request += "%d," % rid
214
215 request = request[:request.__len__() - 1] + ")"
216
217 print(request)
218 cursor.execute(request)
219
220
221
222
223 elif way == 2:
224 print("not random still isn't working")  # TODO add
225 pass
226
227
228 # ---------------------------------- run sql file ----------------------------------
229 def run_sql(addr):
230 sql_file = open(addr, 'r')
231 sql_code = ""
232 for line in sql_file.readlines(): sql_code += line
233 cursor.execute(sql_code)
234
235
236 # ---------------------------------- PARSING METHODS ----------------------------------
237 def store_json(path):
238 """
239 Store dictionary with params to the file by the path.
240 :param path:
241 :return:
242 """
243 params_dict = {'MAX_VCH_LEN': 15,
244 'MIN_PRIOR': 0,
245 'MAX_PRIOR': 2,
246
247 'MIN_PRICE': 20,
248 'MAX_PRICE': 400,
249
250 'MIN_DPRICE': 10,
251 'MAX_DPRICE': 100,
252
253 'MIN_DAY_EXP': 10,
254 'MAX_DAY_EXP': 700,
255
256 'MIN_AM': 1,
257 'MAX_AM': 5,
258
259 'MIN_ADD_AVAIL': 1,
260 'MAX_ADD_AVAIL': 2000
261 }
262 with open(path, "w") as fp:
263 json.dump(params_dict, fp)
```

```
264
265 pass
266
267
268 def parse_json(path):
269 """
270 Load params from json file by path. Json file must contain dictionary.
271 :param path:
272 :return:
273 """
274 print("PARSING PARAMS FROM FILE %s ... " % path, end="")
275
276 params_dict = json.load(open(path, 'r'))
277
278 global MAX_VCH_LEN, MIN_PRIOR, MAX_PRIOR, \
279 MIN_PRICE, MAX_PRICE, \
280 MIN_DPRICE, MAX_DPRICE, \
281 MIN_DAY_EXP, MAX_DAY_EXP, \
282 MIN_AM, MAX_AM, \
283 MIN_WEIGHT, MAX_WEIGHT, \
284 MIN_ADD_AVAIL, MAX_ADD_AVAIL
285
286 MAX_VCH_LEN = params_dict.__getitem__('MAX_VCH_LEN')
287
288 MIN_PRIOR = params_dict.__getitem__('MIN_PRIOR')
289 MAX_PRIOR = params_dict.__getitem__('MAX_PRIOR')
290
291 MIN_PRICE = params_dict.__getitem__('MIN_PRICE')
292 MAX_PRICE = params_dict.__getitem__('MAX_PRICE')
293
294 MIN_DPRICE = params_dict.__getitem__('MIN_DPRICE')
295 MAX_DPRICE = params_dict.__getitem__('MAX_DPRICE')
296
297 MIN_DAY_EXP = params_dict.__getitem__('MIN_DAY_EXP')
298 MAX_DAY_EXP = params_dict.__getitem__('MAX_DAY_EXP')
299
300 MIN_AM = params_dict.__getitem__('MIN_AM')
301 MAX_AM = params_dict.__getitem__('MAX_AM')
302
303 MIN_WEIGHT = params_dict.__getitem__('MIN_WEIGHT')
304 MAX_WEIGHT = params_dict.__getitem__('MAX_WEIGHT')
305
306 MIN_ADD_AVAIL = params_dict.__getitem__('MIN_ADD_AVAIL')
307 MAX_ADD_AVAIL = params_dict.__getitem__('MAX_ADD_AVAIL')
308
309 print("ok")
310
311 pass
312
313
314 #
    ↪ ----------------------------------------------------------------------------------------------------------------------------
    ↪
315 if __name__ == '__main__':
316 if sys.argv.__len__() != 2:
317 raise ValueError("Illegal amount of arguments = %d "
318 "(path to json config file must be passed)" % sys.argv.__len__())
319
320 print("------------------------------------")
321 param_path = sys.argv[1]
```

```
322 # store_json(param_path)
323 parse_json(param_path)
324 print("-----------------------------------")
325
326 login = "refregerator_manager"
327 password = input("Input password for role \'%s\'" % login)
328 print("-----------------------------------")
329
330 conn = psycopg2.connect(dbname='refregerator', user=login, password=password,
        ↪ host='localhost')
331 cursor = conn.cursor()
332 rw = RandomWords()
333
334 fill_complete = False
335 while not fill_complete:
336
337 print("-----------------------------------")
338 # define tables, which will be filled
339 ti = choose_variant_from_dict(
340 "CHOOSE TABLE",
341 {1: 'refregerator', 2: 'product', 3: 'recipe', 4: 'recipe_product', 5: 'exit'}
342 )
343
344 # ---------------------- change table : refregerator
        ↪ ----------------------------------------
345 if ti == 1:
346 add_into_table(
347 cursor, rw, table_name="refregerator",
348 fields={'id': PID, 'product': PREF, 'market_name': PREF, 'price': PINT, '
        ↪ disc_price': PINT,
349 'buying_date': PDATE,
350 'day_before_expiring': PINT, 'amount': PINT},
351 bounds=[(MIN_PRICE, MAX_PRICE), (MIN_DPRICE, MAX_DPRICE), (MIN_DAY_EXP,
        ↪ MAX_DAY_EXP), (MIN_AM, MAX_AM)],
352 min_av=MIN_ADD_AVAIL,
353 max_av=MAX_ADD_AVAIL
354 )
355
356 # ------------------ change tables : product & way_of cooking product
        ↪ ------------------------
357 elif ti == 2:
358 add_into_table(
359 cursor, rw, table_name="product",
360 fields={'id': PID, 'name': PSTR, 'mark': PSTR, 'priority': PINT, 'cook_condition
        ↪ ': PREF,
361 'product_type': PREF},
362 bounds=[(1, MAX_VCH_LEN), (1, MAX_VCH_LEN), (MIN_PRIOR, MAX_PRIOR)],
363 min_av=MIN_ADD_AVAIL,
364 max_av=MAX_ADD_AVAIL
365 )
366
367 # ------------------ change table : recipe ------------------------
368 elif ti == 3:
369 add_into_table(
370 cursor, rw, table_name="recipe",
371 fields={'id': PID, 'name': PSEQ, 'weight':PINT, 'way_of_cooking': PREF},
372 bounds=[(2, 3),(MIN_WEIGHT, MAX_WEIGHT)],
373 min_av=MIN_ADD_AVAIL,
374 max_av=MAX_ADD_AVAIL
375 )
```

```
376
377 # ---------- change table : recipe_product ----------
378 elif ti == 4:
379 add_into_table(
380 cursor, rw, table_name="recipe_product",
381 fields={'id': PID, 'recipe': PREF, 'product': PREF, 'product_amount' : PINT },
382 bounds=[(MIN_AM, MAX_AM)],
383 min_av=MIN_ADD_AVAIL,
384 max_av=MAX_ADD_AVAIL
385 )
386
387 # ---------- exit from program
    ↪ ----------
388 elif ti == 5:
389 fill_complete = True
390
391 # ---------- commit or not commit changes
    ↪ ----------
392 commit_allowed = choose_variant_from_dict(title="COMMIT_CHANGES?", variants={0:
    ↪ 'no', 1: 'yes'})
393 if commit_allowed:
394 conn.commit()
395 cursor.close()
396 conn.close()
```

4. Выводы

В ходе выполнения данной лабораторной работы была написанна параметризированная программа, генерирующая данные для заполнениия базы данных и взаимодействующая с базой данных (добавляет значения в базу).