Санкт-Петербургский Политехнический Университет Петра Великого

Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе №2

Курс: «Базы данных»

Тема: «Создание интерактивного генератора данных»

Выполнил студент:

Бояркин Никита Сергеевич
Группа: 43501/3

Проверил:

Мяснов Александр Владимирович

Санкт-Петербург
2017 г.

Содержание

Лабораторная работа №2

Цель работы

Получить практические навыки работы с БД путем создания собственного интерактивного генератора данных на языке программирования python.

Программа работы

- Разработать интерактивный генератор данных.
- Заполнить таблицы данными с помощью генератора.

Программное окружение

- Python 3.6
- Django 1.10.5
- Psycopg 2.6.2
- PostgreSQL 9.5.6

Ход работы

Создание проекта, конфигурация сервера

Генератор представляет собой management команду, которая принимает два аргумента: название таблицы для добавления и количество данных для генерации.

Данные генерируются для всех таблиц пропорционально их собственному коэффициенту. Например, при генерации в таблице болезней N строк, в таблице лекарств будет сгенерировано $2N$ строк, в таблице поставок будет сгенерировано $4N$ строк и так далее для каждой таблице. Такой метод генерации сохраняет логику базы данных, ведь количество поставок лекарств обычно больше количества лекарств, а количество лекарств обычно больше количества болезней.

Кроме пропорционального заполнения, генератор обеспечивает логическую целостность данных в отдельных таблицах. Это означает, например, что таблицы показаний и противопоказаний не содержат одинаковых пар лекарство + болезнь; таблица несовместимости лекарств не содержит повторяющихся значений; цены, даты и количества лекарств содержатся в определенных границах и др.

Полный листинг команды генерации данных:

```
import random
import string
import time

from datetime import datetime
from django.core.management.base import BaseCommand, CommandError
from DatabaseORM.models import *
from random import randint
from django.core.exceptions import ObjectDoesNotExist, ValidationError
from django.db import DatabaseError

FILE_NAME = 'DatabaseORM/data/'
FILE_EXTENSION = '.data'
```

```
14 DATETIME_FORMAT = '%d.%m.%Y %H:%M:%S'
15
16
17 class Command(BaseCommand):
18     def add_arguments(self, parser):
19         parser.add_argument('table_name', type=str)
20         parser.add_argument('count', type=int)
21
22     @staticmethod
23     def generate_string(length):
24         return ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in
    range(length))
25
26     @staticmethod
27     def generate_double_in_range(minimum, maximum):
28         return random.uniform(minimum, maximum)
29
30     @staticmethod
31     def generate_integer_in_range(minimum, maximum):
32         return randint(minimum, maximum)
33
34     @staticmethod
35     def generate_date_in_range(minimum, maximum):
36         epoch = datetime.utcfromtimestamp(0)
37         minimum_integer = (minimum - epoch).total_seconds()
38         maximum_integer = (maximum - epoch).total_seconds()
39
40         result_integer = Command.generate_integer_in_range(minimum_integer,
    maximum_integer)
41         return datetime.fromtimestamp(result_integer)
42
43     @staticmethod
44     def generate_boolean():
45         return bool(random.getrandbits(1))
46
47     @staticmethod
48     def generate_drug(count):
49         try:
50             id_drug = Drug.objects.latest('id_drug').id_drug
51         except ObjectDoesNotExist:
52             id_drug = -1
53
54         drug_index = 0
55         while drug_index < count:
56             id_drug += 1
57
58             drug_name = None
59             contains = True
60             while contains:
61                 drug_name = Command.generate_string(int(NAME_LENGTH / 16))
62                 contains = bool(Drug.objects.filter(drug_name__contains=drug_name).count
    () > 0)
63
64             drug_shelf_life = Command.generate_integer_in_range(10, 80)
65             drug_current_cost = Command.generate_double_in_range(10, 80)
66             drug_recipe_needed = Command.generate_boolean()
67
68             Drug(id_drug=id_drug, drug_name=drug_name, drug_shelf_life=drug_shelf_life,
    drug_current_cost=drug_current_cost, drug_recipe_needed=drug_recipe_needed).save()
69             drug_index += 1
70
71         Command.generate_disease(int(count / 2))
72
73     @staticmethod
74     def generate_disease(count):
75         try:
```

```
76              id_disease = Disease.objects.latest('id_disease').id_disease
77          except ObjectDoesNotExist:
78              id_disease = -1
79
80          disease_index = 0
81          while disease_index < count:
82              id_disease += 1
83
84              disease_name = None
85              contains = True
86              while contains:
87                  disease_name = Command.generate_string(int(NAME_LENGTH / 16))
88                  contains = bool(Disease.objects.filter(disease_name__contains=
    disease_name).count() > 0)
89
90              Disease(id_disease=id_disease, disease_name=disease_name).save()
91              disease_index += 1
92
93          Command.generate_incompatibility(int(count / 4))
94
95      @staticmethod
96      def generate_incompatibility(count):
97          try:
98              id_incompatibility = Incompatibility.objects.latest('id_incompatibility').
    id_incompatibility
99          except ObjectDoesNotExist:
100             id_incompatibility = -1
101
102         incompatibility_index = 0
103         while incompatibility_index < count:
104             id_incompatibility += 1
105
106             id_drug_first = None
107             id_drug_second = None
108
109             contains = True
110             max_id_drug = Drug.objects.latest('id_drug').id_drug
111             while contains:
112                 id_drug_first = Command.generate_integer_in_range(0, max_id_drug)
113                 id_drug_second = Command.generate_integer_in_range(0, max_id_drug)
114
115                 if id_drug_first == id_drug_second:
116                     continue
117
118                 contains = bool(Drug.objects.filter(id_drug__contains=id_drug_first).
    count() > 0)
119                 contains = contains and bool(Drug.objects.filter(id_drug__contains=
    id_drug_second).count() > 0)
120
121                 if not contains:
122                     contains = True
123                     continue
124                 try:
125                     id_drug_first = Drug.objects.get(id_drug=id_drug_first)
126                     id_drug_second = Drug.objects.get(id_drug=id_drug_second)
127                 except ObjectDoesNotExist:
128                     continue
129
130                 contains = bool(Incompatibility.objects.filter(id_drug_first=
    id_drug_first).filter(id_drug_first=id_drug_second).count() > 0)
131                 contains = contains or bool(Incompatibility.objects.filter(id_drug_first=
    id_drug_second).filter(id_drug_first=id_drug_first).count() > 0)
132
133             try:
134                 Incompatibility(id_incompatibility=id_incompatibility, id_drug_first=
    id_drug_first, id_drug_second=id_drug_second).save()
```

```
135             except ObjectDoesNotExist:
136                 id_incompatibility -= 1
137                 continue
138 
139             incompatibility_index += 1
140 
141         Command.generate_indication(int(count))
142 
143     @staticmethod
144     def generate_indication(count):
145         try:
146             id_indication = Indication.objects.latest('id_indication').id_indication
147         except ObjectDoesNotExist:
148             id_indication = -1
149 
150         indication_index = 0
151         while indication_index < count:
152             id_indication += 1
153 
154             id_disease = None
155             id_drug = None
156 
157             contains = True
158             max_id_disease = Disease.objects.latest('id_disease').id_disease
159             max_id_drug = Drug.objects.latest('id_drug').id_drug
160             while contains:
161                 id_disease = Command.generate_integer_in_range(0, max_id_disease)
162                 id_drug = Command.generate_integer_in_range(0, max_id_drug)
163 
164                 contains = bool(Disease.objects.filter(id_disease__contains=id_disease).
    count() > 0)
165                 contains = contains and bool(Drug.objects.filter(id_drug__contains=
    id_drug).count() > 0)
166 
167                 if not contains:
168                     contains = True
169                     continue
170 
171                 try:
172                     id_disease = Disease.objects.get(id_disease=id_disease)
173                     id_drug = Drug.objects.get(id_drug=id_drug)
174                 except ObjectDoesNotExist:
175                     continue
176 
177                 contains = bool(Indication.objects.filter(id_disease=id_disease).filter(
    id_drug=id_drug).count() > 0)
178                 contains = contains or bool(Contraindication.objects.filter(id_disease=
    id_disease).filter(id_drug=id_drug).count() > 0)
179 
180             try:
181                 Indication(id_indication=id_indication, id_disease=id_disease, id_drug=
    id_drug).save()
182             except ObjectDoesNotExist:
183                 id_indication -= 1
184                 continue
185 
186             indication_index += 1
187 
188         Command.generate_contraindication(int(count))
189 
190     @staticmethod
191     def generate_contraindication(count):
192         try:
193             id_contraindication = Contraindication.objects.latest('id_contraindication').
    id_contraindication
194         except ObjectDoesNotExist:
```

```
            id_contraindication = -1

        contraindication_index = 0
        while contraindication_index < count:
            id_contraindication += 1

            id_disease = None
            id_drug = None

            contains = True
            max_id_disease = Disease.objects.latest('id_disease').id_disease
            max_id_drug = Drug.objects.latest('id_drug').id_drug
            while contains:
                id_disease = Command.generate_integer_in_range(0, max_id_disease)
                id_drug = Command.generate_integer_in_range(0, max_id_drug)

                contains = bool(Disease.objects.filter(id_disease__contains=id_disease).
    count() > 0)
                contains = contains and bool(Drug.objects.filter(id_drug__contains=
    id_drug).count() > 0)

                if not contains:
                    contains = True
                    continue

                try:
                    id_disease = Disease.objects.get(id_disease=id_disease)
                    id_drug = Drug.objects.get(id_drug=id_drug)
                except ObjectDoesNotExist:
                    continue

                contains = bool(Indication.objects.filter(id_disease=id_disease).filter(
    id_drug=id_drug).count() > 0)
                contains = contains or bool(Contraindication.objects.filter(id_disease=
    id_disease).filter(id_drug=id_drug).count() > 0)

            try:
                Contraindication(id_contraindication=id_contraindication, id_disease=
    id_disease, id_drug=id_drug).save()
            except ObjectDoesNotExist:
                id_contraindication -= 1
                continue

            contraindication_index += 1

        Command.generate_provider(int(4 * count))

    @staticmethod
    def generate_provider(count):
        try:
            id_provider = Provider.objects.latest('id_provider').id_provider
        except ObjectDoesNotExist:
            id_provider = -1

        provider_index = 0
        while provider_index < count:
            id_provider += 1

            provider_name = None
            contains = True
            while contains:
                provider_name = Command.generate_string(int(NAME_LENGTH / 16))
                contains = bool(Provider.objects.filter(provider_name__contains=
    provider_name).count() > 0)

            Provider(id_provider=id_provider, provider_name=provider_name).save()
```

```
255             provider_index += 1
256
257         Command.generate_drugstore(int(count))
258
259     @staticmethod
260     def generate_drugstore(count):
261         try:
262             id_drugstore = Drugstore.objects.latest('id_drugstore').id_drugstore
263         except ObjectDoesNotExist:
264             id_drugstore = -1
265
266         drugstore_index = 0
267         while drugstore_index < count:
268             id_drugstore += 1
269
270             drugstore_address = None
271             contains = True
272             while contains:
273                 drugstore_address = Command.generate_string(int(ADDRESS_LENGTH / 16))
274                 contains = bool(Drugstore.objects.filter(drugstore_address__contains=
    drugstore_address).count() > 0)
275
276             Drugstore(id_drugstore=id_drugstore, drugstore_address=drugstore_address).
    save()
277             drugstore_index += 1
278
279         Command.generate_consignment(int(4 * count))
280
281     @staticmethod
282     def generate_consignment(count):
283         try:
284             id_consignment = Consignment.objects.latest('id_consignment').id_consignment
285         except ObjectDoesNotExist:
286             id_consignment = -1
287
288         consignment_index = 0
289         while consignment_index < count:
290             id_consignment += 1
291
292             id_drug = None
293             contains = True
294             max_id_drug = Drug.objects.latest('id_drug').id_drug
295             while contains:
296                 id_drug = Command.generate_integer_in_range(0, max_id_drug)
297                 contains = not bool(Drug.objects.filter(id_drug__contains=id_drug).count
    () > 0)
298                 if not contains:
299                     try:
300                         id_drug = Drug.objects.get(id_drug=id_drug)
301                     except ObjectDoesNotExist:
302                         contains = True
303                         continue
304
305             id_provider = None
306             contains = True
307             max_id_provider = Provider.objects.latest('id_provider').id_provider
308             while contains:
309                 id_provider = Command.generate_integer_in_range(0, max_id_provider)
310                 contains = not bool(Provider.objects.filter(id_provider__contains=
    id_provider).count() > 0)
311                 if not contains:
312                     try:
313                         id_provider = Provider.objects.get(id_provider=id_provider)
314                     except ObjectDoesNotExist:
315                         contains = True
316                         continue
```

```
317
318             id_drugstore = None
319             contains = True
320             max_id_drugstore = Drugstore.objects.latest('id_drugstore').id_drugstore
321             while contains:
322                 id_drugstore = Command.generate_integer_in_range(0, max_id_drugstore)
323                 contains = not bool(Drugstore.objects.filter(id_drugstore__contains=
    id_drugstore).count() > 0)
324                 if not contains:
325                     try:
326                         id_drugstore = Drugstore.objects.get(id_drugstore=id_drugstore)
327                     except ObjectDoesNotExist:
328                         contains = True
329                         continue
330
331             consignment_drug_count = Command.generate_integer_in_range(1000, 10000)
332
333             minimum = datetime(2010, 1, 1)
334             maximum = datetime(2015, 1, 1)
335             while True:
336                 consignment_arrival_date = Command.generate_date_in_range(minimum,
    maximum)
337                 consignment_manufacture_date = Command.generate_date_in_range(minimum,
    maximum)
338
339                 if consignment_arrival_date >= consignment_manufacture_date:
340                     break
341
342             try:
343                 Consignment(id_consignment=id_consignment, id_drug=id_drug, id_provider=
    id_provider, id_drugstore=id_drugstore,
344                             consignment_drug_count=consignment_drug_count,
    consignment_arrival_date=consignment_arrival_date,
345                             consignment_manufacture_date=consignment_manufacture_date).save()
346             except ObjectDoesNotExist:
347                 id_consignment -= 1
348                 continue
349
350             consignment_index += 1
351
352         Command.generate_client(int(count / 16))
353
354     @staticmethod
355     def generate_client(count):
356         try:
357             id_client = Client.objects.latest('id_client').id_client
358         except ObjectDoesNotExist:
359             id_client = -1
360
361         client_index = 0
362         while client_index < count:
363             id_client += 1
364
365             client_name = None
366             contains = True
367             while contains:
368                 client_name = Command.generate_string(int(NAME_LENGTH / 16))
369                 contains = bool(Client.objects.filter(client_name__contains=client_name).
    count() > 0)
370
371             Client(id_client=id_client, client_name=client_name).save()
372             client_index += 1
373
374         Command.generate_request_part(int(2 * count))
375
376     @staticmethod
```

```
377     def generate_request_part(count):
378         try:
379             id_request_part = RequestPart.objects.latest('id_request_part').
    id_request_part
380         except ObjectDoesNotExist:
381             id_request_part = -1
382 
383         request_part_index = 0
384         while request_part_index < count:
385             id_request_part += 1
386 
387             id_client = None
388             contains = True
389             max_id_client = Client.objects.latest('id_client').id_client
390             while contains:
391                 id_client = Command.generate_integer_in_range(0, max_id_client)
392                 contains = not bool(Client.objects.filter(id_client__contains=id_client).
    count() > 0)
393                 if not contains:
394                     try:
395                         id_client = Client.objects.get(id_client=id_client)
396                     except ObjectDoesNotExist:
397                         contains = True
398                         continue
399 
400             request_part_date = Command.generate_date_in_range(datetime(2015, 1, 1),
    datetime(2017, 1, 1))
401 
402             try:
403                 RequestPart(id_request_part=id_request_part, id_client=id_client,
    request_part_date=request_part_date).save()
404             except ObjectDoesNotExist:
405                 id_request_part -= 1
406                 continue
407 
408             request_part_index += 1
409 
410         Command.generate_request(int(4 * count))
411 
412     @staticmethod
413     def generate_request(count):
414         try:
415             id_request = Request.objects.latest('id_request').id_request
416         except ObjectDoesNotExist:
417             id_request = -1
418 
419         request_index = 0
420         while request_index < count:
421             id_request += 1
422 
423             id_request_part = None
424             contains = True
425             max_id_request_part = RequestPart.objects.latest('id_request_part').
    id_request_part
426             while contains:
427                 id_request_part = Command.generate_integer_in_range(0,
    max_id_request_part)
428                 contains = not bool(RequestPart.objects.filter(id_request_part__contains=
    id_request_part).count() > 0)
429                 if not contains:
430                     try:
431                         id_request_part = RequestPart.objects.get(id_request_part=
    id_request_part)
432                     except ObjectDoesNotExist:
433                         contains = True
434                         continue
```

```
435
436             id_consignment = None
437             contains = True
438             max_id_consignment = Consignment.objects.latest('id_consignment').
    id_consignment
439             while contains:
440                 id_consignment = Command.generate_integer_in_range(0, max_id_consignment)
441                 contains = not bool(Consignment.objects.filter(id_consignment__contains=
    id_consignment).count() > 0)
442                 if not contains:
443                     try:
444                         id_consignment = Consignment.objects.get(id_consignment=
    id_consignment)
445                     except ObjectDoesNotExist:
446                         contains = True
447                         continue
448
449             request_drug_previous_cost = Command.generate_double_in_range(10, 80)
450             request_count = Command.generate_integer_in_range(10, 100)
451
452             try:
453                 Request(id_request=id_request, id_request_part=id_request_part,
    id_consignment=id_consignment,
454                         request_drug_previous_cost=request_drug_previous_cost, request_count=
    request_count).save()
455             except ObjectDoesNotExist:
456                     id_request -= 1;
457                     continue
458
459             request_index += 1
460
461     def handle(self, *args, **options):
462         table_name = options['table_name']
463         count = int(options['count'])
464
465         if count < 0:
466             raise CommandError('Wrong count argument.')
467
468         try:
469             if table_name == 'disease':
470                 if count % 4 != 0:
471                     raise CommandError('Wrong count argument.')
472                 self.generate_drug(count * 2)
473             elif table_name == 'drug':
474                 if count % 8 != 0:
475                     raise CommandError('Wrong count argument.')
476                 self.generate_drug(count)
477             elif table_name == 'incompatibility':
478                 self.generate_drug(count * 8)
479             elif table_name == 'indication':
480                 self.generate_drug(count * 8)
481             elif table_name == 'contraindication':
482                 self.generate_drug(count * 8)
483             elif table_name == 'provider':
484                 if count % 4 != 0:
485                     raise CommandError('Wrong count argument.')
486                 self.generate_drug(count * 2)
487             elif table_name == 'drugstore':
488                 if count % 4 != 0:
489                     raise CommandError('Wrong count argument.')
490                 self.generate_drug(count * 2)
491             elif table_name == 'consignment':
492                 if count % 16 != 0:
493                     raise CommandError('Wrong count argument.')
494                 self.generate_drug(int(count / 2))
495             elif table_name == 'client':
```

```
                self.generate_drug(count * 8)
            elif table_name == 'request_part':
                if count % 2 != 0:
                    raise CommandError('Wrong count argument.')
                self.generate_drug(count * 4)
            elif table_name == 'request':
                if count % 8 != 0:
                    raise CommandError('Wrong count argument.')
                self.generate_drug(count)
            else:
                raise CommandError('Wrong table name argument.')
        except (DatabaseError, ValueError, ValidationError) as exception:
            raise CommandError('Unknown exception.')
```

Пример выполнения команды:

```
$ sudo sudo python3.6 manage.py generate disease 1000
```

После этого были сгенерированы данные пропорционально весу каждой таблицы:

- 2000 лекарств.
- 1000 болезней.
- 250 несовместимостей лекарств.
- 250 противопоказаний.
- 250 диагнозов (показаний).
- 250 аптек.
- 250 поставщиков.
- 4000 поставок.
- 250 клиентов.
- 500 частей заказов.
- 2000 заказов.

Вывод

Написание собственного генератора намного более гибкое решение, чем добавление данных вручную. Это обусловлено тем, что при тестировании нас обычно не волнуют точные значения имен, цен и прочих параметров, в то время как пропорции данных между таблицами, контроль повторных значений и неявные зависимости между таблицами важны при тестировании.