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

Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе

Дисциплина: Базы данных

Тема: Создание интерактивного генератора данных

Выполнил студент группы 43501/3	________________ Круминьш Д.В. (подпись)
Преподаватель	________________ Мяснов А.В. (подпись)

Санкт-Петербург
2017

Лабораторная работа

1.1 Цель работы

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

1.2 Ход работы

Была создана команда **generate**, которая имеет два входных параметра:

1. **tableName** - название таблицы или области для которой необходимо сгенерировать данные. В случае ввода **all** будет генерация для всех таблиц.
2. **count** - целочисленное число, обозначающие количество строк, которые необходимо сгенерировать.

Также есть оптиональный параметр:

1. **-f** - в случае добавления параметра, данные будут генерироваться случайным образом, а не путем взятия случайных строк из заранее подготовленных текстовых файлов.

Необходимые, для генерации, данные берутся из заранее созданных файлов с соответствующим содержанием. Далее приведен список этих файлов:

- names.txt
- surnames.txt
- oldnames.txt
- manufacturers.txt
- country.txt
- property.txt
- properties.txt
- type.txt
- product_name.txt
- product_description.txt
- review_description.txt
- address.txt
- phone.txt

Например формирование ФИО клиента будет происходить путем взятия случайных строк из файлов **names.txt, surnames.txt, oldnames.txt**.

Наиболее интересной таблицей для генерации является таблица **Type**, представленная на рисунке 1.1.

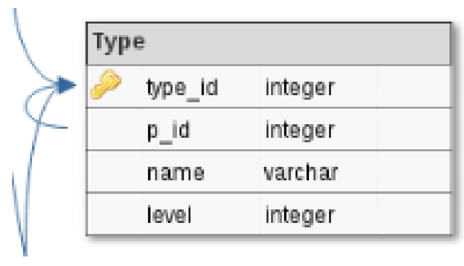
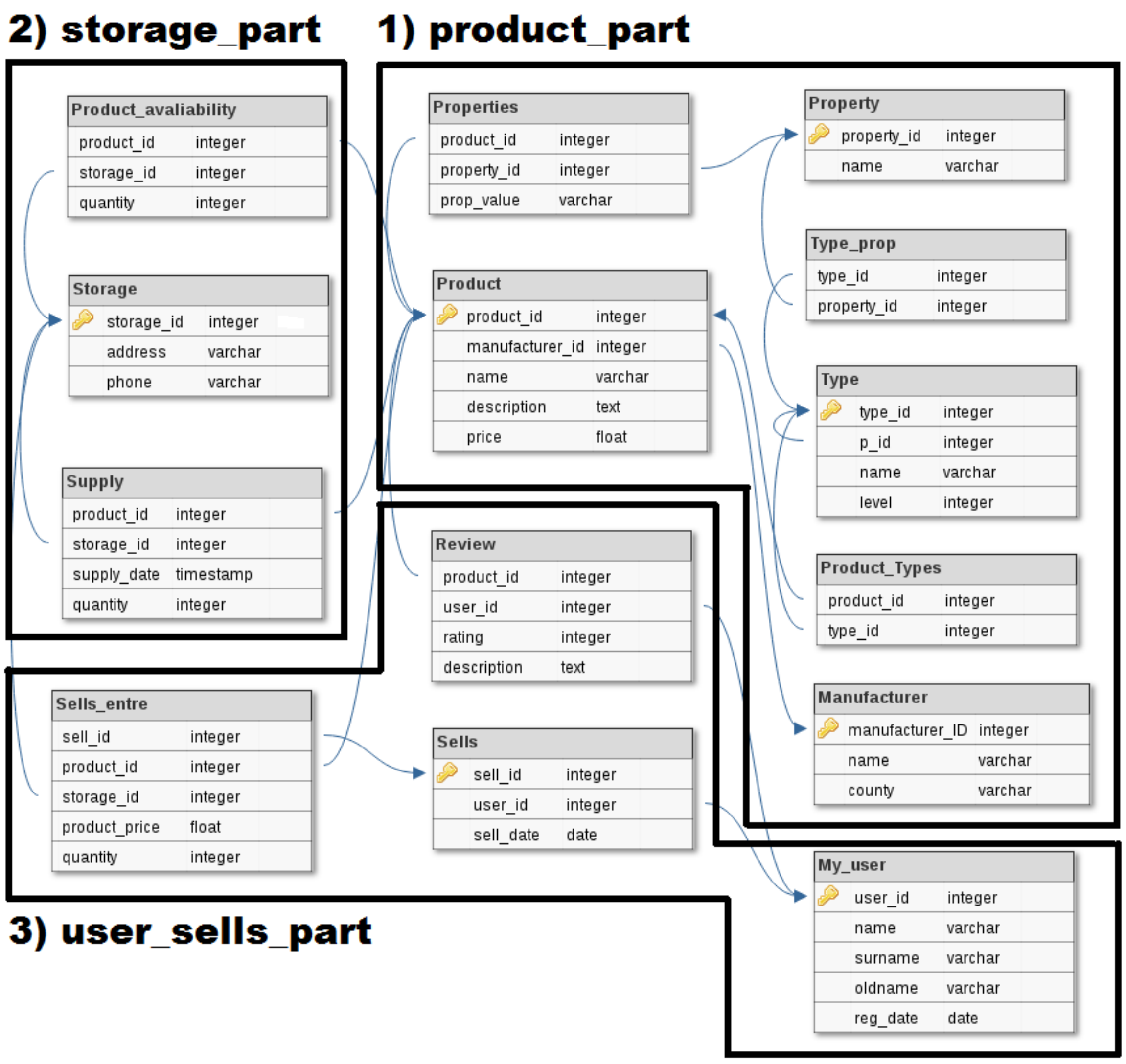


Рис. 1.1: Таблица Type

Если в работах прошлого семестра, поле level не использовалось, то в данной работе, с помощью него контролируется глубина вложенности. Так при генерации новой записи, у родительского элемента(если таковой имеется) проверяется значение поля level. Таким образом можно задать максимальную вложенность.

Так-же имеется разделенность, схемы базы данных, по областям в соответствии с их контентом.



2) storage_part
1) product_part
3) user_sells_part
Product_avaliability: product_id integer, storage_id integer, quantity integer
Storage: storage_id integer, address varchar, phone varchar
Supply: product_id integer, storage_id integer, supply_date timestamp, quantity integer
Properties: product_id integer, property_id integer, prop_value varchar
Property: property_id integer, name varchar
Type_prop: type_id integer, property_id integer
Product: product_id integer, manufacturer_id integer, name varchar, description text, price float
Type: type_id integer, p_id integer, name varchar, level integer
Product_Types: product_id integer, type_id integer
Review: product_id integer, user_id integer, rating integer, description text
Manufacturer: manufacturer_ID integer, name varchar, county varchar
Sells_entre: sell_id integer, product_id integer, storage_id integer, product_price float, quantity integer
Sells: sell_id integer, user_id integer, sell_date date
My_user: user_id integer, name varchar, surname varchar, oldname varchar, reg_date date

Рис. 1.2: Разделенная на области схема БД

Для каждой из областей также задается параметр count, однако для вложенных таблиц его коэффициент будет несколько изменяться.

1. product_part
   - Product -> count*1
   - Manufacturer -> count*2
   - Property -> count*20
   - Properties -> count*10
   - Type -> count*4
   - Type_prop -> count*10
   - Product_types -> count*2
2. storage_part
   - Storage -> count*1
   - Product_avaliability -> count*10
   - Supply -> count*10
3. user_sells_part
   - My_user -> count
   - Sells -> count*5
   - Sells_entre -> count*10
   - Review -> count*10

Далее приведен пример использования команды, для генерации 10 новых строчек в каждую из таблиц базы данных.

```
C:\study\s08\BD\task_2\work\lab_1>python manage.py generate all 10
10 row(s) successfully added in table my_user.
10 row(s) successfully added in table manufacturer.
10 row(s) successfully added in table sells.
10 row(s) successfully added in table property.
10 row(s) successfully added in table type.
10 row(s) successfully added in table type_prop.
10 row(s) successfully added in table product.
10 row(s) successfully added in table product_types.
10 row(s) successfully added in table review.
10 row(s) successfully added in table storage.
10 row(s) successfully added in table product_avaliability.
10 row(s) successfully added in table properties.
10 row(s) successfully added in table supply.
10 row(s) successfully added in table sells_entre.
```

Листинг 1.1: Пример использования команды

Код команды **generate** представлен в приложении 1.

1.3 Вывод

В ходе данной работы было продолжено создание собственного приложения, которое работает с базой данных. В частности был написан собственный генератор данных. В отличии от встроенных генераторов в какие-либо СУБД, в данном случае генератор в конечном итоге получается более гибким, который можно как-либо изменять.

Хоть и генератор является гибким, у него имеются некоторые проблемы с производительностью, время генерации данных у него заметно выше чем при использовании СУБД. Скорее всего это вызвано тем, что многие данных для записи в таблицы считываются из файлов, что замедляет генерацию.

Так-же на время генерации влияют и некоторые проверки на возможность генерации данных. Например имеются ли в таблице, которая связана с текущей по вторичному ключу, данные и если есть то какой диапазон id имеется в данной таблице.

Приложение 1

```
from django.core.management.base import BaseCommand
from django.db.models import Max, Min
from ulmart.models import *
import random
import datetime
import string
import argparse

NAMES='Data/names.txt'
SURNAMES='Data/surnames.txt'
OLDNAMES='Data/oldnames.txt'
MANUFACTURERS='Data/manufacturers.txt'
COUNTRY='Data/country.txt'
PROPERTY='Data/property.txt'
PROPERTIES='Data/properties.txt'
TYPE='Data/type.txt'
PRODUCT_NAME='Data/product_name.txt'
PRODUCT_DESCRIPTION='Data/product_description.txt'
REVIEW_DESCRIPTION='Data/review_description.txt'
ADDRESS='Data/address.txt'
PHONE='Data/phone.txt'

MAXIMUM_LEVEL=3

class Command(BaseCommand):
    def add_arguments(self, parser):
        parser.add_argument('table', type=str)
        parser.add_argument('count', type=int)
        parser.add_argument("-f","--fromFile", action="store_true")

    def getLinesCount(self, filename):
        with open(filename, 'r') as f:
            return(sum(1 for _ in f))

    def getRandomLine(self, filename):
        #Random int between 0 and line's count
        num=random.randint(0,self.getLinesCount(filename)-1)

        #Opening file, and searching for needed line
        f = open(filename, 'r')
        i=0
        for line in f:
            if i==num:
                return(str.strip(line))
            i+=1
        return("null")

    def getRandomString(self):
        return(''.join(random.choice(string.ascii_uppercase + string.
    ↪ digits) for _ in range(10)))

    def addUsers(self, count, fromFile):
        #Counter
        added=0

        #Check if this table is empty
        if My_user.objects.count()==0:
            max_id=0
        else:
            max_id = My_user.objects.order_by('-user_id')[0].user_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            if fromFile:
                new_name=self.getRandomLine(NAMES)
                new_surname=self.getRandomLine(SURNAMES)
                new_oldname=self.getRandomLine(OLDNAMES)
            else:
                new_name=self.getRandomString()
                new_surname=self.getRandomString()
                new_oldname=self.getRandomString()
            new_date=datetime.date(random.randint(2006,2016), random.randint(1,12),random.randint(1,28))

            #Creating new object and saving it
            try:
                new_user = My_user(user_id=new_id, name=new_name, surname=new_surname, oldname=new_oldname, reg_date=new_date)
                new_user.save()
                added=added+1
            except:
                print('Error while trying add new row.')
            i+=1
        print(str(added)+" row(s) added in table my_user.")

    def addSells(self, count):
        #Counter
        added=0

        #Check if there is no users
        if My_user.objects.count()==0:
            print('No users!')
            return

        #Check if this table is empty
        if Sells.objects.count()==0:
            max_id=0
        else:
            max_id = Sells.objects.order_by('-sell_id')[0].sell_id

        #Variables for generation limits
        min_user_id=My_user.objects.order_by('user_id')[0].user_id
        max_user_id=My_user.objects.order_by('-user_id')[0].user_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            new_user_id=random.randint(min_user_id, max_user_id)
            new_date=datetime.date(random.randint(2006,2016), random.
   ↪ randint(1,12),random.randint(1,28))

            #Creating new object and saving it
            try:
                new_sell = Sells(sell_id=new_id, user_id=new_user_id,
   ↪ sell_date=new_date)
                new_sell.save()
                added=added+1
            except:
                print('Error while trying add new row.')
            i+=1
        print(str(added)+" row(s) added in table sells.")

    def addManufacturers(self, count, fromFile):
        #Counter
        added=0

        #Check if this table is empty
        if Manufacturer.objects.count()==0:
            max_id=0
        else:
            max_id = Manufacturer.objects.order_by('-manufacturer_id')
   ↪ [0].manufacturer_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            if fromFile:
                new_name=self.getRandomLine(MANUFACTURERS)
                new_country=self.getRandomLine(COUNTRY)
            else:
                new_name=self.getRandomString()
                new_country=self.getRandomString()

            #Creating new object and saving it
            try:
                new_manufacturer = Manufacturer(manufacturer_id=new_id,
   ↪ name=new_name, country=new_country)
                new_manufacturer.save()
                added=added+1
            except:
                print('Error while trying add new row.')
            i+=1
        print(str(added)+" row(s) added in table manufacturer.")

    def addProperty(self, count, fromFile):
        #Counter
        added=0

        #Check if this table is empty
        if Property.objects.count()==0:
            max_id=0
        else:
            max_id = Property.objects.order_by('-property_id')[0].property_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            if fromFile:
                new_name=self.getRandomLine(PROPERTY)
            else:
                new_name=self.getRandomString()

            #Creating new object and saving it
            try:
                new_property = Property(property_id=new_id, name=new_name)
                new_property.save()
                added=added+1
            except:
                print('Error while trying add new row.')

            i+=1
        print(str(added)+" row(s) added in table property.")

    def addType_prop(self, count):
        #Counter
        added=0

        #Check if there is no data in property or type
        if Property.objects.count()==0 or Type.objects.count()==0:
            print('No data in property or type table!')
            return

        #Check if this table is empty
        if Type_prop.objects.count()==0:
            max_id=0
        else:
            max_id = Type_prop.objects.order_by('-id')[0].id

        #Variables for generation limits
        min_property_id=Property.objects.order_by('property_id')[0].property_id
        max_property_id=Property.objects.order_by('-property_id')[0].property_id

        min_type_id=Type.objects.order_by('type_id')[0].type_id
        max_type_id=Type.objects.order_by('-type_id')[0].type_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            new_type_id=random.randint(min_type_id, max_type_id)
            new_property_id=random.randint(min_property_id, max_property_id)

            #Creating new object and saving it
            try:
                new_type_prop = Type_prop(id=new_id, property_id=new_property_id, type_id=new_type_id)
                new_type_prop.save()
                added=added+1
            except:
                print('Error while trying add new row.')

            i+=1
        print(str(added)+" row(s) added in table type_prop.")

    def addSells_entre(self, count):
        #Counter
        added=0

        #Check if there is no data in storage or product
        if Storage.objects.count()==0 or Product.objects.count()==0 or Sells.objects.count()==0:
            print('No data in storage or product or sells table!')
            return

        #Check if this table is empty
        if Sells_entre.objects.count()==0:
            max_id=0
        else:
            max_id = Sells_entre.objects.order_by('-id')[0].id

        #Variables for generation limits
        min_product_id=Product.objects.order_by('product_id')[0].product_id
        max_product_id=Product.objects.order_by('-product_id')[0].product_id

        min_storage_id=Storage.objects.order_by('storage_id')[0].storage_id
        max_storage_id=Storage.objects.order_by('-storage_id')[0].storage_id

        min_sell_id=Sells.objects.order_by('sell_id')[0].sell_id
        max_sell_id=Sells.objects.order_by('-sell_id')[0].sell_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            new_sell_id=random.randint(min_sell_id, max_sell_id)
            new_product_id=random.randint(min_product_id, max_product_id
   ↪ )
            new_storage_id=random.randint(min_storage_id, max_storage_id
   ↪ )
            new_product_price=random.uniform(1000, 40000)
            new_quantity=random.randint(1,100)

            #Creating new object and saving it
            try:
                new_sells_entre = Sells_entre(id=new_id, sell_id=
   ↪ new_sell_id, product_id=new_product_id, storage_id=new_storage_id,
   ↪  product_price=new_product_price, quantity=new_quantity)
                new_sells_entre.save()
                added=added+1
            except:
                print('Error while trying add new row.')

            i+=1
        print(str(added)+" row(s) added in table sells_entre.")

    def addSupply(self, count):
        #Counter
        added=0

        #Check if there is no data in storage or product
        if Storage.objects.count()==0 or Product.objects.count()==0:
            print('No data in storage or product table!')
            return

        #Check if this table is empty
        if Supply.objects.count()==0:
            max_id=0
        else:
            max_id = Supply.objects.order_by('-id')[0].id

        #Variables for generation limits
        min_product_id=Product.objects.order_by('product_id')[0].
   ↪ product_id
        max_product_id=Product.objects.order_by('-product_id')[0].
   ↪ product_id

        min_storage_id=Storage.objects.order_by('storage_id')[0].
   ↪ storage_id
        max_storage_id=Storage.objects.order_by('-storage_id')[0].
   ↪ storage_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            new_product_id=random.randint(min_product_id, max_product_id)
            new_storage_id=random.randint(min_storage_id, max_storage_id)
            new_supply_date=datetime.date(random.randint(2006,2016), random.randint(1,12),random.randint(1,28))
            new_quantity=random.randint(1,500)

            #Creating new object and saving it
            try:
                new_supply = Supply(id=new_id, product_id=new_product_id, storage_id=new_storage_id, supply_date=new_supply_date, quantity=new_quantity)
                new_supply.save()
                added=added+1
            except:
                print('Error while trying add new row.')

            i+=1
        print(str(added)+" row(s) added in table supply.")

    def addProperties(self, count, fromFile):
        #Counter
        added=0

        #Check if there is no data in property or product
        if Property.objects.count()==0 or Product.objects.count()==0:
            print('No data in property or product table!')
            return

        #Check if this table is empty
        if Properties.objects.count()==0:
            max_id=0
        else:
            max_id = Properties.objects.order_by('-id')[0].id

        #Variables for generation limits
        min_property_id=Property.objects.order_by('property_id')[0].property_id
        max_property_id=Property.objects.order_by('-property_id')[0].property_id

        min_product_id=Product.objects.order_by('product_id')[0].product_id
        max_product_id=Product.objects.order_by('-product_id')[0].product_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            new_product_id=random.randint(min_product_id, max_product_id )
            new_property_id=random.randint(min_property_id, max_property_id)
            if fromFile:
                new_prop_value=self.getRandomLine(PROPERTIES)
            else:
                new_prop_value=self.getRandomString()

            #Creating new object and saving it
            try:
                new_properties = Properties(id=new_id, product_id=new_product_id, property_id=new_property_id, prop_value=new_prop_value)
                new_properties.save()
                added=added+1
            except:
                print('Error while trying add new row.')

            i+=1
        print(str(added)+" row(s) added in table properties.")

    def addProduct_avaliability(self, count):
        #Counter
        added=0

        #Check if there is no data in storage or product
        if Storage.objects.count()==0 or Product.objects.count()==0:
            print('No data in storage or product table!')
            return

        #Check if this table is empty
        if Product_avaliability.objects.count()==0:
            max_id=0
        else:
            max_id = Product_avaliability.objects.order_by('-id')[0].id

        #Variables for generation limits
        min_product_id=Product.objects.order_by('product_id')[0].product_id
        max_product_id=Product.objects.order_by('-product_id')[0].product_id

        min_storage_id=Storage.objects.order_by('storage_id')[0].storage_id
        max_storage_id=Storage.objects.order_by('-storage_id')[0].storage_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            new_product_id=random.randint(min_product_id, max_product_id
   ↪ )
            new_storage_id=random.randint(min_storage_id, max_storage_id
   ↪ )
            new_quantity=random.randint(1,500)

            #Creating new object and saving it
            try:
                new_product_avaliability = Product_avaliability(id=
   ↪ new_id, product_id=new_product_id, storage_id=new_storage_id,
   ↪ quantity=new_quantity)
                new_product_avaliability.save()
                added=added+1
            except:
                print('Error while trying add new row.')
            i+=1
        print(str(added)+" row(s) added in table product_avaliability.")

    def addProduct_types(self, count):
        #Counter
        added=0

        #Check if there is no data in product or type
        if Product.objects.count()==0 or Type.objects.count()==0:
            print('No data in product or type table!')
            return

        #Check if this table is empty
        if Product_types.objects.count()==0:
            max_id=0
        else:
            max_id = Product_types.objects.order_by('-id')[0].id

        #Variables for generation limits
        min_product_id=Product.objects.order_by('product_id')[0].
   ↪ product_id
        max_product_id=Product.objects.order_by('-product_id')[0].
   ↪ product_id

        min_type_id=Type.objects.order_by('type_id')[0].type_id
        max_type_id=Type.objects.order_by('-type_id')[0].type_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            new_type_id=random.randint(min_type_id, max_type_id)
            new_product_id=random.randint(min_product_id, max_product_id
   ↪ )

            #Creating new object and saving it
            try:
                new_product_type = Product_types(id=new_id, product_id=
   ↪ new_product_id, type_id=new_type_id)
                new_product_type.save()
                added=added+1
            except:
                print('Error while trying add new row.')

            i+=1
        print(str(added)+" row(s) added in table product_types.")

    def addProduct(self, count, fromFile):
        #Counter
        added=0

        #Check if there is no data in manufacturer
        if Manufacturer.objects.count()==0:
            print('No data in manufacturer table!')
            return

        #Check if this table is empty
        if Product.objects.count()==0:
            max_id=0
        else:
            max_id = Product.objects.order_by('-product_id')[0].
   ↪ product_id

        #Variables for generation limits
        min_manufacturer_id=Manufacturer.objects.order_by('
   ↪ manufacturer_id')[0].manufacturer_id
        max_manufacturer_id=Manufacturer.objects.order_by('-
   ↪ manufacturer_id')[0].manufacturer_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            new_manufacturer_id=random.randint(min_manufacturer_id,
   ↪ max_manufacturer_id)
            if fromFile:
                new_name=self.getRandomLine(PRODUCT_NAME)
                new_description=self.getRandomLine(PRODUCT_DESCRIPTION)
            else:
                new_name=self.getRandomString()
                new_description=self.getRandomString()
            new_price=random.uniform(1000, 40000)

            #Creating new object and saving it
            try:
                new_product = Product(product_id=new_id, manufacturer_id
   ↪ =new_manufacturer_id, name=new_name, description=new_description,
   ↪ price=new_price)
                new_product.save()
                added=added+1
```

```
            except:
                print('Error while trying add new row.')

            i+=1
        print(str(added)+" row(s) added in table product.")

    def addStorage(self, count, fromFile):
        #Counter
        added=0

        #Check if this table is empty
        if Storage.objects.count()==0:
            max_id=0
        else:
            max_id = Storage.objects.order_by('-storage_id')[0].
↪ storage_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            if fromFile:
                new_address=self.getRandomLine(ADDRESS)
                new_phone=self.getRandomLine(PHONE)
            else:
                new_address=self.getRandomString()
                new_phone=self.getRandomString()

            #Creating new object and saving it
            try:
                new_storage = Storage(storage_id=new_id, address=
↪ new_address, phone=new_phone)
                new_storage.save()
                added=added+1
            except:
                print('Error while trying add new row.')

            i+=1
        print(str(added)+" row(s) added in table storage.")

    def addReview(self, count, fromFile):
        #Counter
        added=0

        #Check if there is no data in product or my_user
        if Product.objects.count()==0 or My_user.objects.count()==0:
            print('No data in product or my_user table!')
            return

        #Check if this table is empty
        if Review.objects.count()==0:
            max_id=0
        else:
            max_id = Review.objects.order_by('-id')[0].id

        #Variables for generation limits
        min_product_id=Product.objects.order_by('product_id')[0].product_id
        max_product_id=Product.objects.order_by('-product_id')[0].product_id

        min_my_user_id=My_user.objects.order_by('user_id')[0].user_id
        max_my_user_id=My_user.objects.order_by('-user_id')[0].user_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            new_product_id=random.randint(min_product_id, max_product_id)
            new_user_id=random.randint(min_my_user_id, max_my_user_id)
            new_rating=random.randint(1,5)
            if fromFile:
                new_description=self.getRandomLine(REVIEW_DESCRIPTION)
            else:
                new_description=self.getRandomString()

            #Creating new object and saving it
            try:
                new_review = Review(id=new_id, product_id=new_product_id, user_id=new_user_id, rating=new_rating, description=new_description)
                new_review.save()
                added=added+1
            except:
                print('Error while trying add new row.')

            i+=1
        print(str(added)+" row(s) added in table review.")

    def addType(self, count, fromFile):
        #Counter
        added=0

        #Check if this table is empty
        if Type.objects.count()==0:
            max_id=0
        else:
            max_id = Type.objects.order_by('-type_id')[0].type_id

        #Starting of loop
        i=1
        while i<=count:
            new_id=max_id+i
            if fromFile:
                new_name=self.getRandomLine(TYPE)
            else:
                new_name=self.getRandomString()

            #50 at 50 if new type will have parent, also checking if parent is possible
            if random.randint(0,1)==0 or new_id==1:
                new_p_id=None
                new_level=1
            else:
                #Random parent
                min_p_id=Type.objects.order_by('type_id')[0].type_id
                max_p_id=Type.objects.order_by('-type_id')[0].type_id
                new_p_id=random.randint(min_p_id, max_p_id)
                #Selecting parent level and incrasing it
                new_level=Type.objects.get(pk=new_p_id).level+1
                if new_level>MAXIMUM_LEVEL:
                    continue

            #Creating new object and saving it
            try:
                new_type = Type(type_id=new_id, p_id=new_p_id, name=new_name, level=new_level)
                new_type.save()
                added=added+1
            except:
                print('Error while trying add new row.')

            i+=1
        print(str(added)+" row(s) added in table type.")

    def handle(self, *args, **options):
        #Reading input options
        table = options['table']
        count = int(options['count'])

        #Checking of options
        if count<=0:
            print('Wrong count!')
            return
        if table=='my_user':
            self.addUsers(count, options['fromFile'])
        elif table=='sells':
            self.addSells(count)
        elif table=='manufacturer':
            self.addManufacturers(count, options['fromFile'])
        elif table=='property':
            self.addProperty(count, options['fromFile'])
        elif table=='type':
            self.addType(count, options['fromFile'])
        elif table=='type_prop':
            self.addType_prop(count)
        elif table=='supply':
            self.addSupply(count)
        elif table=='product':
            self.addProduct(count, options['fromFile'])
        elif table=='product_types':
            self.addProduct_types(count)
        elif table=='review':
            self.addReview(count, options['fromFile'])
        elif table=='properties':
            self.addProperties(count, options['fromFile'])
        elif table=='sells_entre':
            self.addSells_entre(count)
        elif table=='storage':
            self.addStorage(count, options['fromFile'])
        elif table=='product_avaliability':
            self.addProduct_avaliability(count)
        elif table=='product_part':
            self.addManufacturers(count*2, options['fromFile'])
            self.addProduct(count, options['fromFile'])
            self.addProperty(count*20, options['fromFile'])
            self.addProperties(count*10, options['fromFile'])
            self.addType(count*4, options['fromFile'])
            self.addType_prop(count*10)
            self.addProduct_types(count*2)
        elif table=='storage_part':
            self.addStorage(count, options['fromFile'])
            self.addProduct_avaliability(count*10)
            self.addSupply(count*10)
        elif table=='user_sells_part':
            self.addUsers(count, options['fromFile'])
            self.addSells(count*5)
            self.addReview(count*10, options['fromFile'])
            self.addSells_entre(count*10)
        elif table=='all':
            self.addUsers(count, options['fromFile'])
            self.addManufacturers(count, options['fromFile'])
            self.addSells(count)
            self.addProperty(count, options['fromFile'])
            self.addType(count, options['fromFile'])
            self.addType_prop(count)
            self.addProduct(count, options['fromFile'])
            self.addProduct_types(count)
            self.addReview(count, options['fromFile'])
            self.addStorage(count, options['fromFile'])
            self.addProduct_avaliability(count)
            self.addProperties(count, options['fromFile'])
            self.addSupply(count)
            self.addSells_entre(count)
```

Листинг 1.2: generate.py