Университет ИТМО

Факультет Инфокоммуникационных технологий

Направление подготовки «09.03.03 Мобильные и сетевые технологии»

Лабораторная работа №4

### **ЗАПРОСЫ НА ВЫБОРКИ ДАННЫХ К БД POSTGRESQL. ПРЕДСТАВЛЕНИЯ В POSTGRESQL.**

Выполнила:

Егоров М. П.

Группа К3240

Преподаватель:

Говорова М. М.

Санкт-Петербург

2020

**Цель работы**:

Овладеть практическими навыками создания представлений и запросов на выборку данных к базе данных PostgreSQL и использования подзапросов при модификации данных.

**Практическое задание:**

1. Создать базу данных с использованием Django.
2. Создать схему в составе базы данных.
3. Создать таблицы базы данных.
4. Установить ограничения на данные.
5. Заполнить таблицы БД рабочими данными.
6. Создать резервную копию БД.
7. Восстановить БД.

**Выполнение:**

1. **Название создаваемой БД**

«LotosLab»

1. **Django databases**

Сменим со стандартной базы данных Django SQLite на PostgreSQL. Установим нужные зависимости:

$ pip install psycopg2-binary

Запустим PostregSQL и создадим базу данных lotos, добавим пользователя Django, установим соответствующие роли:

$ sudo -u postgres psql postgres

$ create user django with password 'Password-123';

$ alter role django set client\_encoding to 'utf8';

$ alter role django set default\_transaction\_isolation to 'read committed';

$ alter role django set timezone to 'UTC';

$ create database lotos owner django;

Далее заменим в файле settings.py базу данных:

DATABASES = {

'default': {

'HOST': '127.0.0.1:8000',

'ENGINE': 'django.db.backends.postregsql,

'NAME': 'lotos',

'USER': 'django',

'PASSWORD': 'Password-123',

'OPTIONS': {

'init\_command': "SET sql\_mode='STRICT\_TRANS\_TABLES'",

'charset': 'utf8mb4',

},

}

}

1. **Django ORM**

Определим схемы через ORM фреймворка Python – Django. Нужно определить название таблицы и ее атрибуты, а также процедуры. В языке Python это соответственно название класса, название и типы атрибутов (django.fields), методы класса.

Листинг 1 – Моделирование базы данных с помощью ORM Django

class City(models.Model):

name = models.CharField(max\_length=127)

declination = models.CharField(max\_length=127)

latitude = models.FloatField()

longitude = models.FloatField()

tapable = models.BooleanField(default=True)

def \_\_str\_\_(self):

return self.reversed\_name

@property

def reversed\_name(self):

transilted = translit(self.name, 'ru', reversed=True).lower()

return re.sub('[^a-z0-9-]', '', transilted)

def get\_city\_by\_name(name):

for city in City.objects.all():

if city.reversed\_name == name:

return city

def shops\_geojson(self):

return {

'type': 'FeatureCollection',

'features': [{

'type': 'Feature',

'geometry': {

'type': 'Point',

'coordinates': shop.coords,

},

'properties': {

'id': shop.id,

'city\_name': self.name,

'lab\_name': shop.labaratory.name,

'shop\_name': shop.name,

'address': f"{shop.address}",

'take\_text': shop.parse\_take\_text(),

'color': 'red',

}

} for lab in LabaratoryBranch.objects.filter(city=self)

for shop in LabaratoryBranchShop.objects.filter(labaratory=lab)]

}

class SamplingType(models.Model):

name = models.CharField(max\_length=255)

site\_name = models.CharField(max\_length=255)

def \_\_str\_\_(self):

return translit(self.name, 'ru', reversed=True)

def from\_dict(dictionary):

sampling = SamplingType()

for field\_name, val in dictionary.items():

setattr(sampling, field\_name, val)

return sampling

def create\_lab\_branch\_sampling(self, price):

for lab in LabaratoryBranch.objects.all():

SamplingInLabaratoryBranch.objects.create(

labaratory=lab,

sampling\_type=self,

price=price

)

class GeneralLabaratory(models.Model):

name = models.CharField(max\_length=63)

email = models.EmailField()

is\_active = models.BooleanField(default=True)

def \_\_str\_\_(self):

return translit(self.name, 'ru', reversed=True)

class LabaratoryBranch(models.Model):

labaratory = models.ForeignKey('GeneralLabaratory', on\_delete=models.CASCADE)

city = models.ForeignKey('City', on\_delete=models.CASCADE)

is\_tapable = models.BooleanField(default=True)

def \_\_str\_\_(self):

return f"{self.city} @{self.labaratory}"

@property

def name(self):

return self.labaratory.name

@property

def is\_active(self):

return self.labaratory.is\_active

class LabaratoryBranchShop(models.Model):

name = models.CharField(max\_length=127)

labaratory = models.ForeignKey('LabaratoryBranch', on\_delete=models.CASCADE)

address = models.CharField(max\_length=127)

telephone = models.CharField(max\_length=127)

latitude = models.FloatField()

longitude = models.FloatField()

def \_\_str\_\_(self):

return f"{self.labaratory} @{translit(self.name, 'ru', reversed=True)}"

def set\_coodrs(self, latitude, longitude):

self.latitude = latitude

self.longitude = longitude

@property

def coords(self):

return self.longitude, self.latitude

@property

def gen\_lab(self):

return self.labaratory.labaratory

def parse\_take\_text(self):

today = timezone.now()

if self.labaratory.id == 3:

if self.labaratory.city.id == 1:

if today.hour < 13 and today.weekday() <= 4:

delta = 1

elif today.hour >= 13 and today.weekday() <= 3:

delta = 2

elif today.weekday() > 3:

delta = 8 - today.weekday()

else:

if today.weekday() == 4:

delta = 4

elif today.hour > 13 and today.weekday() == 3:

delta = 4

elif today.hour < 13 and today.weekday() <= 3:

delta = 2

elif today.hour >= 13 and today.weekday() <= 3:

delta = 3

elif today.weekday() > 3:

delta = 9 - today.weekday()

next\_date = today + datetime.timedelta(days=delta)

month\_name = next\_date.strftime('%B')

if month\_name in self.scl:

month\_name = self.scl[month\_name]

text = f'Вы можете сдать анализы с {next\_date.day} {month\_name} в течение двух недель'

else:

text = "Вы можете сдать анализы с завтрашнего дня в течение двух недель"

return text;

scl = {

'Январь': 'января',

'Февраль': 'февраля',

'Март': 'марта',

'Апрель': 'апреля',

'Май': 'майя',

'Июнь': 'июня',

'Июль': 'июля',

'Август': 'августа',

'Сентябрь': 'сентября',

'Октябрь': 'октября',

'Ноябрь': 'ноября',

'Декабрь': 'декабря',

}

class SamplingInLabaratoryBranch(models.Model):

labaratory = models.ForeignKey(LabaratoryBranch, on\_delete=models.CASCADE)

sampling\_type = models.ForeignKey(SamplingType, on\_delete=models.CASCADE)

price = models.IntegerField(default=0)

def \_\_str\_\_(self):

return f"{self.sampling\_type} @{self.labaratory}"

class Analys(models.Model):

labaratory = models.ForeignKey('GeneralLabaratory', on\_delete=models.CASCADE)

code = models.CharField(max\_length=63)

name = models.TextField()

section = models.CharField(max\_length=255)

lab\_section = models.CharField(max\_length=255)

description = models.CharField(max\_length=1023, blank=True)

preparations = models.TextField(blank=True)

research\_type = models.TextField(blank=True)

blank\_form = models.CharField(max\_length=63, blank=True)

synonyms\_rus = models.TextField()

synonyms\_eng = models.TextField()

sampling\_type = models.ManyToManyField(SamplingType, blank=True)

image = models.ImageField(upload\_to='analysis', default='analysis/default.jpg')

def \_\_str\_\_(self):

return f"code: {self.code} @{self.name\_en}"

@property

def name\_en(self):

transilted = translit(self.name, 'ru', reversed=True).replace(' ', '-').lower()

return re.sub('[^a-z0-9-]', '', transilted)

@property

def matched(self):

return Matched.find\_matched\_by\_code(self.code)

@property

def matched\_id(self):

matched = self.matched

if not matched is None:

return matched.id

return -1

@property

def synonym\_rus\_list(self):

return self.synonyms\_rus.split(',')

@property

def synonym\_eng\_list(self):

return self.synonyms\_eng.split(',')

def get\_analys\_by\_name\_en(name):

for analys in Analys.objects.all():

if analys.name\_en == name:

return analys

def price\_by\_city(self, city):

return AnalysInBranch.objects.filter(analys=self, labaratory\_branch\_city=city)

def get\_sections\_list():

default\_lab = GeneralLabaratory.objects.get(name='Инвитро')

sections\_list = set([analys.section.capitalize() for analys in Analys.objects.filter(labaratory=default\_lab)])

return list(sections\_list)

def from\_dict(row):

analys = Analys.objects.filter(code=row['code']).first()

if analys is None:

analys = Analys()

labaratory = GeneralLabaratory.objects.filter(name=row['lab\_name']).first()

if labaratory is None:

return f'Лаборатории {row["lab\_name"]} нет'

fields = ['name', 'code', 'lab\_section', 'section', 'description',

'preparations', 'research\_type', 'blank\_form',

'synonyms\_eng', 'synonyms\_rus']

for field in fields:

if field in row and not pandas.isna(row[field]):

setattr(analys, field, row[field])

analys.labaratory = labaratory

return analys

class AnalysInBranch(models.Model):

labaratory\_branch = models.ForeignKey('LabaratoryBranch', on\_delete=models.CASCADE)

analys = models.ForeignKey('Analys', on\_delete=models.CASCADE)

lab\_price = models.IntegerField()

our\_price = models.IntegerField(default=0)

period = models.TextField()

def \_\_str\_\_(self):

return f"{self.analys} @{self.labaratory\_branch}"

@property

def name(self):

return self.analys.name

@property

def price(self):

return int((self.lab\_price - self.our\_price) \* 0.75 + self.our\_price)

@property

def is\_buyable(self):

return self.our\_price != 0 and self.labaratory\_branch.labaratory.is\_active and self.labaratory\_branch.is\_tapable

@property

def image(self):

return self.analys.image

@property

def labaratory\_name(self):

return self.labaratory.name

@property

def city(self):

return self.labaratory\_branch.city

@property

def city\_name(self):

return self.city.name

@property

def matched(self):

return Matched.find\_matched\_by\_code(self.analys.code)

@property

def matched\_analysis(self):

matched = self.matched

if not matched is None:

return matched.analysis\_by\_city(self.labaratory\_branch.city)

return [self]

@property

def matched\_labaratories(self):

return [analys.labaratory\_branch.name for analys in self.matched\_analysis]

@property

def matched\_id(self):

matched = self.matched

if not matched is None:

return matched.id

return -1

@property

def minimal\_price(self):

matched = self.matched\_analysis

if not matched is None:

return min([analys.price for analys in matched])

return self.price

def matched\_labaratories\_by\_id(self, id):

if id == -1:

return self.labaratory\_branch.name

matched = Matched.objects.get(id=id)

matched\_analysis = matched.analysis\_by\_city(self.labaratory\_branch.city)

return [analys.labaratory\_branch.name for analys in matched\_analysis]

def get\_samplings(self):

samplings\_objects = SamplingInLabaratoryBranch.objects.filter(

sampling\_type\_\_in=self.analys.sampling\_type.all(),

labaratory=self.labaratory\_branch

)

samplings = []

for sampling in samplings\_objects:

samplings.append({

'id': sampling.id,

'sampling': sampling.sampling\_type,

'price': sampling.price,

})

return samplings

def from\_dict(row):

analys\_branch = AnalysInBranch.objects.filter(

analys\_\_code=row['code'],

labaratory\_branch\_\_labaratory\_\_name=row['lab\_name'],

labaratory\_branch\_\_city\_\_name=row['city\_name']).first()

if analys\_branch is None:

analys\_branch = AnalysInBranch()

analys = Analys.from\_dict(row)

if isinstance(analys, str):

return analys

analys.save()

city = City.objects.filter(name=row['city\_name']).first()

if city is None:

return f'Города {row["city\_name"]} нет в базе'

labaratory = LabaratoryBranch.objects.filter(

labaratory=analys.labaratory, city=city).first()

if labaratory is None:

return f'в городе {city.name} нет лаборатории {row["lab\_name"]}'

analys\_branch.labaratory\_branch = labaratory

analys\_branch.analys = analys

fields = ['lab\_price', 'our\_price']

for field in fields:

if field in row and not pandas.isna(row[field]):

setattr(analys\_branch, field, row[field])

analys\_branch.save()

return analys\_branch

class Matched(models.Model):

codes = models.TextField()

def \_\_str\_\_(self):

return self.codes

def find\_matched\_by\_code(code):

for matched in Matched.objects.all():

if code in matched.codes.split(';'):

return matched

@property

def default\_analys\_name(self):

name = ""

for code in self.codes.split(';'):

analys = Analys.objects.filter(code=code).first()

if not analys is None:

name = analys.name

if analys.labaratory.name == settings.DEFAULT\_LABORATORY\_FOR\_ANALYS\_NAME:

return name

return name

def analysis(self):

analysis = []

for code in self.codes.split(';'):

analys = Analys.objects.filter(code=code).first()

analysis.append(analys)

return analysis

def analysis\_by\_city(self, city):

analysis = self.analysis()

branch\_analysis = AnalysInBranch.objects.filter(analys\_\_in=analysis, labaratory\_branch\_\_city=city)

return branch\_analysis

def from\_dict(dictionary):

matched = Matched()

for field\_name, val in dictionary.items():

setattr(matched, field\_name, val)

return matched

Процедура: получить партнеров из определенной специализации. Соответствует запросу «SELECT \* FROM Partner WHERE specialization\_en = spec»:

Text

Description automatically generated with medium confidence

Рисунок 2 - Определение метода сущности «Партнер»

Остальные сущности находятся в файле «models.py».

1. **Миграции базы данных и выгрузка базы данных**

После того, как все сущности были определены на языке Python нужно сделать миграции, чтобы модели были реализованы в PostgreSQL.

После миграции нужно добавить данные из csv файлов: списки анализов, списки магазинов и партнеров, реализованы функции UPDATE и INSERT.

Листинг 2 - Функции заполнения данных из csv файлов

def import\_matcheds(request):

Matched.objects.all().delete()

df = pandas.read\_csv(settings.BASE\_DIR + f'/export\_data/csv/matched.csv')

for series\_row in df.iloc:

matched = Matched.from\_dict(series\_row.to\_dict())

matched.save()

return HttpResponse(f"Добавлено {Matched.objects.all().count()} сопоставлений")

def import\_samplings(request):

df = pandas.read\_csv(settings.BASE\_DIR + f'/export\_data/csv/samplings.csv')

for series\_row in df.iloc:

sampling = SamplingType.from\_dict(series\_row.to\_dict())

sampling.save()

return HttpResponse(f"Добавлено {SamplingType.objects.all().count()} сопоставлений")

def refresh\_analys\_sections(request):

df = pandas.read\_csv(settings.BASE\_DIR + f'/export\_data/csv/sections\_connections.csv')

df = df.fillna('ДРУГОЕ')

d = {}

for i, h, c in df.iloc:

d[i] = i

d[h] = i

d[c] = i

unexist = set()

for analys in Analys.objects.all():

if not analys.lab\_section in d:

unexist.add(analys.lab\_section)

continue

analys.section = d[analys.lab\_section]

analys.save()

if analys.id == 46797:

print(analys, analys.section)

return HttpResponse('Успешно изменено')

def import\_shops(request):

LabaratoryBranchShop.objects.all().delete()

for city\_name in os.listdir(settings.BASE\_DIR + f'/export\_data/Locations'):

city = City.get\_city\_by\_name(city\_name)

if city is None:

continue

for filename in os.listdir(settings.BASE\_DIR + f'/export\_data/Locations/' + city\_name):

if filename[-3:] != 'csv':

continue

df = pandas.read\_csv(settings.BASE\_DIR + f'/export\_data/Locations/{city\_name}/{filename}', sep='\t')

for row in df.iloc:

lab = LabaratoryBranch.objects.filter(city=city,

labaratory\_\_name=row['lab\_name']).first()

if lab is None:

labaratory = GeneralLabaratory.objects.filter(

name=row['lab\_name']

).first()

if labaratory is None:

labaratory = GeneralLabaratory.objects.create(

name=row['lab\_name']

)

lab = LabaratoryBranch.objects.create(

city=city,

labaratory=labaratory,

)

LabaratoryBranchShop.objects.create(

labaratory=lab,

address=row['address'],

latitude=row['latitude'],

longitude=row['longitude'],

name=lab.name,

telephone=''

)

return HttpResponse('Успешно!')

def refresh\_analysis(self):

df = pandas.read\_csv(settings.BASE\_DIR + f'/export\_data/analysis/analysis.csv', sep=';')

errors = []

for row in df.iloc:

try:

analys = AnalysInBranch.from\_dict(row)

if isinstance(analys, str):

return HttpResponse(analys)

except Exception as e:

errors.append(e)

return HttpResponse('Успешно' if len(errors) == 0 else str(errors))

Как только данные загрузятся, можно сделать выгрузку данных с помощью инструментов Python в формат json:

$ python manage.py dumpdata > lotos.json

1. **Запросы на базу данных**

Одной из ключевых задач является получение информации об анализе: в зависимости от города просмотра анализа может иметь аналоги, нужно их тоже показать на странице. Реализованы запросы SELECT.

Листинг 3 – запрос получения информации об анализе по названию города и названию анализа.

def analys(request, city\_name, name\_en):

city = City.get\_city\_by\_name(city\_name)

analys = Analys.get\_analys\_by\_name\_en(name\_en)

branch\_analys = AnalysInBranch.objects.filter(analys=analys, labaratory\_branch\_\_city=city).first()

if branch\_analys is None:

return redirect('/eror')

matcheds = Matched.find\_matched\_by\_code(branch\_analys.analys.code)

if not matcheds is None:

analys\_list = matcheds.analysis\_by\_city(city)

labaratory\_list = [analys.labaratory\_branch for analys in analys\_list]

else:

analys\_list = [branch\_analys]

labaratory\_list = [branch\_analys.labaratory\_branch]

title = branch\_analys.name

return render(request, 'analysis/analys.html', locals())

1. **Восстановление данных**

Чтобы восстановить данные нужно ввести команду:

$ python manage.py loaddata lotos.json

**Выводы:**

Была реализована инфологическая модель «LotosLab» в реляционную базу данных PostgreSQL с помощью фреймворка Django. Были загружены данные из csv файлов и сделана выгрузка всей базы для восстановления.