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:

I. Название создаваемой БД

«LotosLab»

II. 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',
```

```
        },
      }
    }
```

III. 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)
```

```python
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»:

```
def get_partners_by_spec_name(spec):
    partners = Partner.objects.filter(specialization_en=spec)

    return partners
```

Рисунок 2 - Определение метода сущности «Партнер»

Остальные сущности находятся в файле «models.py».

IV. 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
```

V. 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())
```

VI. 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.