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: Знакомство с ORM на примере Django

Выполнил студент группы 43501/3 _____ Круминьш Д.В.
(подпись)

Преподаватель _____ Мяснов А.В.
(подпись)

Санкт-Петербург
2017

1 Цель работы

Получить практические навыки работы с БД через механизм объектно-реляционного отображения.

2 Программа работы

1. Знакомство с фреймворком Django:
   - установка
   - создание проекта
   - создание приложения
2. Формирование набора моделей, соответствующих схеме БД, полученной по результатам разработки схемы БД и модификации схемы
3. Знакомство с механизмом миграций: автоматическое формирование схемы БД с помощью миграций
4. Создание manage-команд для заполнения БД тестовыми (по несколько записей в каждой таблице)
5. Написание отчета

3 Ход работы

3.1 Подготовка

Предварительно был установлен, следующий комплекс программ:

- Python 3.6;
- PostgreSQL 9.6.2;
- Psycopg 2.6.2;
- Django 1.10.5.

Для PostgreSQL была создана база данных **ulmart**, а также пользователь **psaer**. Далее был создан проект, следующими командами:

```
1 C:\study\s08\БД\task_1\work>django-admin.py startproject lab_1
2 C:\study\s08\БД\task_1\work>cd lab_1
3 C:\study\s08\БД\task_1\work\lab_1>python manage.py startapp ulmart
```

Листинг 1: Some Code

Для изменения стандартных настроек, был открыт файл ...work\lab_1\lab_1\settings.py. Первоначально в нем содержались следующие строчки:

```
1 ...
2
3 DATABASES = {
4     'default': {
5         'ENGINE': 'django.db.backends.sqlite3',
```

```
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

...

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

...
```

Листинг 2: Some Code

Которые были заменены на следующие:

```
...

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'ulmart',
        'USER': 'psaer',
        'PASSWORD': '1234',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

...

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ulmart',
]

...
```

Листинг 3: Some Code

Далее, для формирования моделей необходимо открыть файл ...work\lab_1\ulmart\models.py. Именно в данном файле, необходимо описать модели, для последующей миграции.

3.2 Миграция

По результатам прошлых лабораторных работ, имеется схема базы-данных, приведенная на рис. 1, согласно которой, будут формироваться соответствующие модели для миграции.

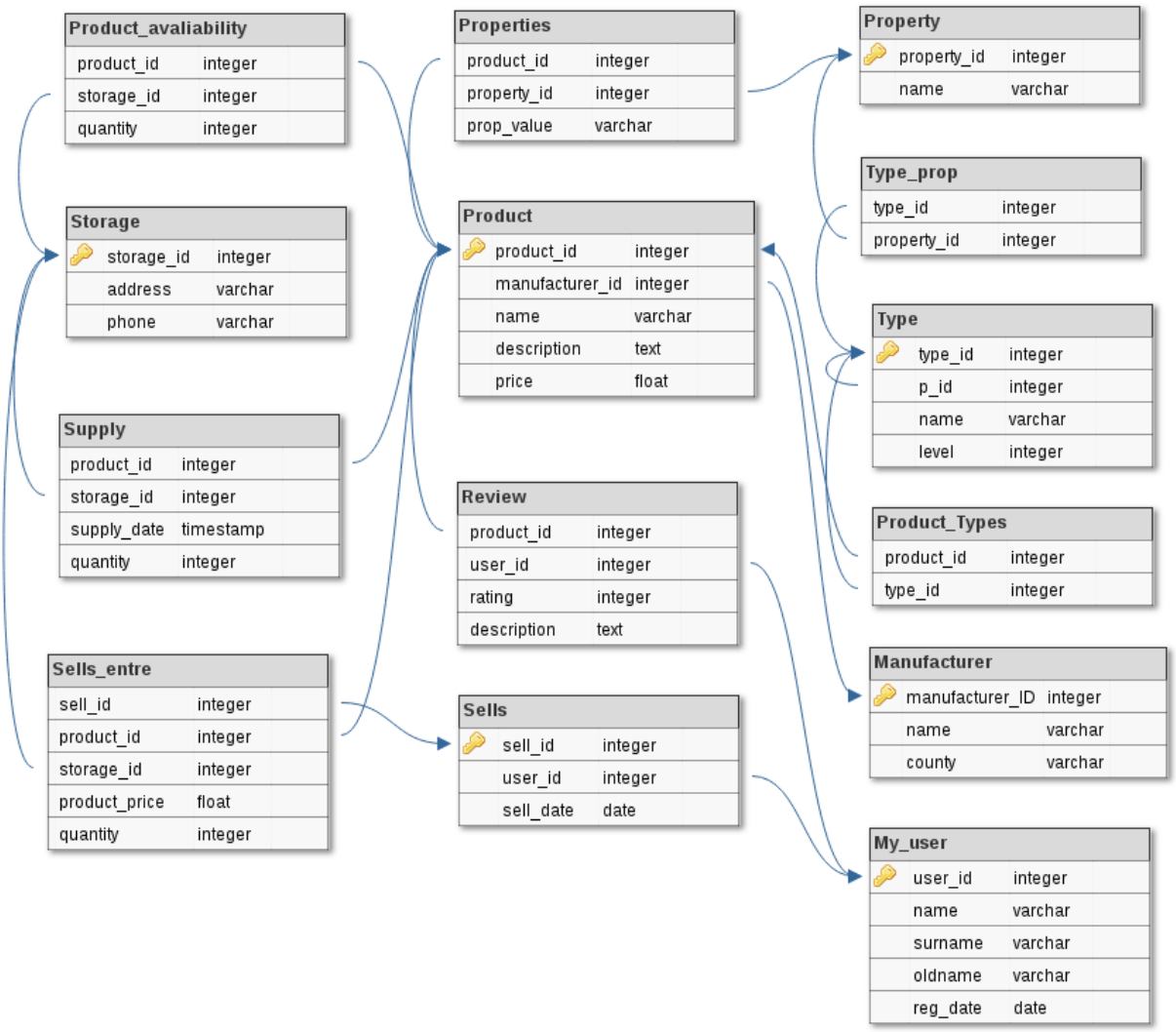

Рис. 1: SQL-схема БД

В файл models.py были добавлены модели, соответствующие приведенной базе данных.

```
1  from django.db import models
2
3  class My_user(models.Model):
4      user_id=models.IntegerField(primary_key=True)
5      name=models.CharField(max_length=200)
6      surname=models.CharField(max_length=200)
7      oldname=models.CharField(max_length=200)
8      reg_date = models.DateField(auto_now=False, auto_now_add=True)
9
10     class Meta:
11         db_table = "My_user"
12
13 class Sells(models.Model):
14     sell_id=models.IntegerField(primary_key=True)
15     user=models.ForeignKey('My_user')
```

```
16      sell_date = models.DateField(auto_now=False, auto_now_add=False)
17
18      class Meta:
19          db_table = "Sells"
20
21  class Manufacturer(models.Model):
22      manufacturer_id=models.IntegerField(primary_key=True)
23      name=models.CharField(max_length=200)
24      country=models.CharField(max_length=200)
25
26      class Meta:
27          db_table = "Manufacturer"
28
29  class Type(models.Model):
30      type_id=models.IntegerField(primary_key=True)
31      p=models.ForeignKey('Type', null=True)
32      name=models.CharField(max_length=200)
33      level=models.IntegerField()
34
35      class Meta:
36          db_table = "Type"
37
38  class Property(models.Model):
39      property_id=models.IntegerField(primary_key=True)
40      name=models.CharField(max_length=200)
41
42      class Meta:
43          db_table = "Property"
44
45  class Type_prop(models.Model):
46      type=models.ForeignKey('Type')
47      property=models.ForeignKey('Property')
48
49      class Meta:
50          db_table = "Type_prop"
51
52  class Product(models.Model):
53      product_id=models.IntegerField(primary_key=True)
54      manufacturer=models.ForeignKey('Manufacturer')
55      name=models.CharField(max_length=200)
56      description=models.CharField(max_length=200)
57      price=models.FloatField()
58
59      class Meta:
60          db_table = "Product"
61
62  class Product_types(models.Model):
63      product=models.ForeignKey('Product')
64      type=models.ForeignKey('Type')
65
66      class Meta:
67          db_table = "Product_types"
68
```

```
class Properties(models.Model):
    product=models.ForeignKey('Product')
    property=models.ForeignKey('Property')
    prop_value=models.CharField(max_length=200)

    class Meta:
        db_table = "Properties"

class Review(models.Model):
    product=models.ForeignKey('Product')
    user=models.ForeignKey('My_user')
    rating=models.IntegerField()
    description=models.CharField(max_length=200)

    class Meta:
        db_table = "Review"

class Storage(models.Model):
    storage_id=models.IntegerField(primary_key=True)
    address=models.CharField(max_length=200)
    phone=models.CharField(max_length=200)

    class Meta:
        db_table = "Storage"

class Product_avaliability(models.Model):
    product=models.ForeignKey('Product')
    storage=models.ForeignKey('Storage')
    quantity=models.IntegerField()

    class Meta:
        db_table = "Product_avaliability"

class Supply(models.Model):
    product=models.ForeignKey('Product')
    storage=models.ForeignKey('Storage')
    supply_date = models.DateField(auto_now=False, auto_now_add=True)
    quantity=models.IntegerField()

    class Meta:
        db_table = "Supply"

class Sells_entre(models.Model):
    sell=models.ForeignKey('Sells')
    product=models.ForeignKey('Product')
    storage=models.ForeignKey('Storage')
    product_price=models.FloatField()
    quantity=models.IntegerField()

    class Meta:
        db_table = "Sells_entre"
```

Листинг 4: models.py

Примечание: если ключ является вторичным, то Django автоматически допишет **_id** к данному полю. Также необходимо указывать класс Meta для каждой таблицы, иначе Django автоматически допишет название проекта к каждой таблице. После написания моделей был запущен процесс миграции.

```
C:\study\s08\БД\task_1\work\lab_1>python manage.py makemigrations
   ↪ ulmart
Migrations for 'ulmart':
ulmart\migrations\0001_initial.py:
- Create model Manufacturer
- Create model My_user
- Create model Product
- Create model Product_avaliability
- Create model Product_types
- Create model Properties
- Create model Property
- Create model Review
- Create model Sells
- Create model Sells_entre
- Create model Storage
- Create model Supply
- Create model Type
- Create model Type_prop
- Add field storage to sells_entre
- Add field property to properties
- Add field type to product_types
- Add field storage to product_avaliability

C:\study\s08\БД\task_1\work\lab_1>python manage.py migrate ulmart
Operations to perform:
Apply all migrations: ulmart
Running migrations:
Applying ulmart.0001_initial... OK
```

Листинг 5: Процесс миграция

По завершению миграции, база данных содержит все таблицы из схемы, а также таблицу **django_migrations**, которая необходима для работы системы миграции Django.

3.3 Manage-команды

Для реализации manage-команд по заполнению базы данных, в директорию проекта **ulmart** была добавлена директория **management** и сопутствующие файлы. Дерево директории ulmart, теперь выглядит следующим образом:

```
ulmart
├── admin.py
├── __init__.py
├── management
│   ├── commands
│   │   ├── __init__.py
│   │   └── populate_db.py
│   └── __init__.py
├── models.py
└── прочие файлы
```

Файл **populate_db.py** как раз и отвечает за исполнение команд. Для заполнения каждой таблицы 3 значениями, в данный файл были внесены соответствующие команды.

```
from django.core.management.base import BaseCommand
from ulmart.models import *

class Command(BaseCommand):
    args = '<foo bar ...>'
    help = 'our help string comes here'

    def _create_property(self):
        temp1 = Property(property_id=1, name='prop_1')
        temp1.save()
        temp2 = Property(property_id=2, name='prop_2')
        temp2.save()
        temp3 = Property(property_id=3, name='prop_3')
        temp3.save()

    def _create_type(self):
        temp1 = Type(type_id=1, name='type_1', level=1)
        temp1.save()
        temp2 = Type(type_id=2, p_id=1, name='type_2', level=2)
        temp2.save()
        temp3 = Type(type_id=3, name='type_3', level=1)
        temp3.save()

    def _create_type_prop(self):
        temp1 = Type_prop(id=1, property_id=1, type_id=1)
        temp1.save()
        temp2 = Type_prop(id=2, property_id=2, type_id=2)
        temp2.save()
        temp3 = Type_prop(id=3, property_id=3, type_id=3)
        temp3.save()

    def _create_manufacturer(self):
        temp1 = Manufacturer(manufacturer_id=1, name='manufac_1',
    ↪ country='China')
        temp1.save()
        temp2 = Manufacturer(manufacturer_id=2, name='manufac_2',
    ↪ country='China')
        temp2.save()
        temp3 = Manufacturer(manufacturer_id=3, name='manufac_3',
    ↪ country='China')
        temp3.save()

    def _create_product(self):
        temp1 = Product(product_id=1, manufacturer_id=1, name='name_1'
    ↪ , description='desc_1', price=5000.20)
        temp1.save()
        temp2 = Product(product_id=2, manufacturer_id=2, name='name_2'
    ↪ , description='desc_2', price=8000.20)
        temp2.save()
        temp3 = Product(product_id=3, manufacturer_id=3, name='name_3'
    ↪ , description='desc_3', price=9000.20)
```

```
46          temp3.save()
47
48      def _create_product_types(self):
49          temp1 = Product_types(id=1, product_id=1, type_id=1)
50          temp1.save()
51          temp2 = Product_types(id=2, product_id=2, type_id=2)
52          temp2.save()
53          temp3 = Product_types(id=3, product_id=3, type_id=3)
54          temp3.save()
55
56      def _create_properties(self):
57          temp1 = Properties(id=1, product_id=1, property_id=1,
    ↪ prop_value='value_1')
58          temp1.save()
59          temp2 = Properties(id=2, product_id=2, property_id=3,
    ↪ prop_value='value_2')
60          temp2.save()
61          temp3 = Properties(id=3, product_id=3, property_id=3,
    ↪ prop_value='value_3')
62          temp3.save()
63
64      def _create_my_user(self):
65          temp1 = My_user(user_id=1, name='name_1', surname='surname_1',
    ↪  oldname='oldname_1', reg_date='2017-10-10')
66          temp1.save()
67          temp2 = My_user(user_id=2, name='name_2', surname='surname_2',
    ↪  oldname='oldname_2', reg_date='2017-10-10')
68          temp2.save()
69          temp3 = My_user(user_id=3, name='name_3', surname='surname_3',
    ↪  oldname='oldname_3', reg_date='2017-10-10')
70          temp3.save()
71
72      def _create_review(self):
73          temp1 = Review(id=1, product_id=1, user_id=1, rating=5,
    ↪ description='desc_1')
74          temp1.save()
75          temp2 = Review(id=2, product_id=2, user_id=2, rating=5,
    ↪ description='desc_2')
76          temp2.save()
77          temp3 = Review(id=3, product_id=3, user_id=3, rating=5,
    ↪ description='desc_3')
78          temp3.save()
79
80      def _create_sells(self):
81          temp1 = Sells(sell_id=1, user_id=1, sell_date='2017-10-10')
82          temp1.save()
83          temp2 = Sells(sell_id=2, user_id=2, sell_date='2017-10-10')
84          temp2.save()
85          temp3 = Sells(sell_id=3, user_id=3, sell_date='2017-10-10')
86          temp3.save()
87
88      def _create_storage(self):
89          temp1 = Storage(storage_id=1, address='addr_1', phone='phone_1
```

```
    ↪ ')
90          temp1.save()
91          temp2 = Storage(storage_id=2, address='addr_2', phone='phone_2
    ↪ ')
92          temp2.save()
93          temp3 = Storage(storage_id=3, address='addr_3', phone='phone_3
    ↪ ')
94          temp3.save()
95
96      def _create_product_avaliability(self):
97          temp1 = Product_avaliability(id=1, product_id=1, storage_id=1,
    ↪  quantity=10)
98          temp1.save()
99          temp2 = Product_avaliability(id=2, product_id=2, storage_id=2,
    ↪  quantity=10)
100         temp2.save()
101         temp3 = Product_avaliability(id=3, product_id=3, storage_id=3,
    ↪  quantity=10)
102         temp3.save()
103
104     def _create_supply(self):
105         temp1 = Supply(id=1, product_id=1, storage_id=1, supply_date='
    ↪ 2017-10-10', quantity=5)
106         temp1.save()
107         temp2 = Supply(id=2, product_id=2, storage_id=2, supply_date='
    ↪ 2017-10-10', quantity=5)
108         temp2.save()
109         temp3 = Supply(id=3, product_id=3, storage_id=3, supply_date='
    ↪ 2017-10-10', quantity=5)
110         temp3.save()
111
112     def _create_sells_entre(self):
113         temp1 = Sells_entre(id=1, sell_id=1, product_id=1, storage_id
    ↪ =1, product_price=413.21, quantity=10)
114         temp1.save()
115         temp2 = Sells_entre(id=2, sell_id=2, product_id=3, storage_id
    ↪ =2, product_price=442.21, quantity=10)
116         temp2.save()
117         temp3 = Sells_entre(id=3, sell_id=3, product_id=3, storage_id
    ↪ =3, product_price=5313.21, quantity=10)
118         temp3.save()
119
120
121     def handle(self, *args, **options):
122         self._create_property()
123         self._create_type()
124         self._create_type_prop()
125         self._create_manufacturer()
126         self._create_product()
127         self._create_product_types()
128         self._create_properties()
129         self._create_my_user()
130         self._create_review()
```

```
131         self._create_sells()
132         self._create_storage()
133         self._create_product_avaliability()
134         self._create_supply()
135         self._create_sells_entre()
```

Листинг 6: populate_db.py

Теперь запустим команду и проверим с помощью psql.

```
1  C:\study\s08\БД\task_1\work\lab_1>python manage.py populate_db
2
3  C:\study\s08\БД\task_1\work\lab_1>psql -U postgres
4  Пароль пользователя postgres:
5  psql (9.6.2)
6  Введите "help", чтобы получить справку.
7
8  postgres=# \c ulmart
9  Вы подключены к базе данных "ulmart" как пользователь "postgres".
10 ulmart=# select * from "Product";
11 product_id  |  name  | description | price  | manufacturer_id
12 ------------+--------+-------------+--------+-----------------
13 1           | name_1 | desc_1      | 5000.2 |               1
14 2           | name_2 | desc_2      | 8000.2 |               2
15 3           | name_3 | desc_3      | 9000.2 |               3
```

Листинг 7: Выполнение manage-команды

Чего и требовалось ожидать, команда успешно добавила данные в каждую из таблиц.

4 Вывод

В результате данной работы было проведено знакомство с миграциями моделей используя Django, а также с manage-командами для наполнения базы данных. К достоинствам миграции Django можно отнести:

- Ускорение процесса изменения схемы базы данных;
- Возможность отслеживания схемы базы данных;
- Поддержка многими бэкендами(PostgreSQL, MySQL, SQLite);
- Возможность отката.

Несмотря на подобные достоинства, я бы все равно предпочел-бы проводить какие-либо операции с базами данных используя прямые sql запросы и команды.

Также был проведен опыт использования manage-команд. Использование данного инструмента позволяет расширить проект, написанием каких-либо собственных команд. Так например, можно написать генератор для заполнения полей таблиц в базе данных.