CS459 Django web framework 2018

Wasit Limprasert, PhD, wasit_l@sci.tu.ac.th
Department of Computer Science, Faculty of Science and Technology, Thammasat University

$ git clone https://github.com/wasit7/cs459_django2018.git

Week01 First Run Server

```
cd cs459_django2018
mkdir week01
django-admin startproject project01
cd project01
python manage.py runserver
git add -A
git commit -m "first run server"
```

Add Superuser

```
python manage.py makemigrations
python manage.py migrate
python manage.py createsuperuser
```

go to (localhost:8000/admin)[localhost:8000/admin]

reference:
https://github.com/wasit7/cs459_django2018/tree/master/week01

Week02 From previous week

First Run Server

```
cd cs459_django2018
mkdir week01
django-admin startproject project01
cd project01
python manage.py runserver
git add -A
git commit -m "first run server"
```

Add Superuser

```
python manage.py makemigrations
python manage.py migrate
python manage.py createsuperuser
```

go to (localhost:8000/admin)[localhost:8000/admin]

For this week

Install python packages

```
pip install -r requirement.txt
```

create app

```
python manage.py startapp myapp
```

install app /myproject/setting.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp',
    'crispy_forms',
    'django_extensions',
]
```

setup media storage and crispy form

```
CRISPY_TEMPLATE_PACK = 'bootstrap3'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'
```

edit model.py

```
from __future__ import unicode_literals
from django.db import models

class Person(models.Model):
    name=models.CharField(max_length=100)
    dob=models.DateField(blank=True,null=True)
    def __unicode__(self):
        return u"%s"%(self.name)

class Image(models.Model):
    image=models.ImageField(upload_to='images')
    description=models.CharField(max_length=100,blank=True,null=True)
```

view.py

This file work with /myapp/templates/*

```
from django.views.generic.edit import CreateView, UpdateView
from django.views.generic.list import ListView
from .forms import PersonForm
from .models import Person, Image

from django.shortcuts import render
```

```python
def home(request):
        return render(request, 'home.html', {'key': "value" })

class CreatePersonView(CreateView):
        queryset = Person()
        template_name='person.html'
        form_class = PersonForm
        success_url = '/'

class UpdatePersonView(UpdateView):
        queryset = Person.objects.all()
        template_name='person.html'
        form_class = PersonForm
        success_url = '/'

class ListPersonView(ListView):
    model = Person
    template_name='person_list.html'
```

forms.py

```python
from django import forms
from django.forms import ModelForm
from crispy_forms.helper import FormHelper
from crispy_forms.layout import Submit
#from django.forms.extras.widgets import SelectDateWidget
from django.contrib.admin import widgets
import datetime
from .models import Person

class PersonForm(ModelForm):
        class Meta:
                model =  Person
                exclude=[]

                widgets = {
                        'dob': forms.DateInput(
                                attrs={
                                'type': 'date',
                'value': datetime.datetime.now().strftime("%Y-%m-%d")
                }, format="%Y-%m-%d"
            ),
        }

    def __init__(self, *args, **kwargs):
        super(PersonForm, self).__init__(*args, **kwargs)
        self.helper = FormHelper()
        self.helper.add_input(Submit('submit', 'Submit'))
```

admin.py

```
from django.contrib import admin
from myapp.models import Person, Image

class PersonAdmin(admin.ModelAdmin):
    list_display=[f.name for f in Person._meta.fields]

admin.site.register(Person,PersonAdmin)

class ImageAdmin(admin.ModelAdmin):
    list_display=[f.name for f in Image._meta.fields]

admin.site.register(Image,ImageAdmin)
```

migrate and run

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver
```

Reference:
https://github.com/wasit7/cs459_django2018/tree/master/week02/week02_project

Week03 basic web view-url

views.py

this file define the responses of requests

```
from django.shortcuts import render
from django.http import HttpResponse
import datetime

def current_datetime(request):
    now = datetime.datetime.now()
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(html)
```

urls.py

managing the routing of request from an url to a view function

```
from django.conf.urls import url, include
from django.contrib import admin
from . import views

urlpatterns = [
    url(r'^current_datetime/', views.current_datetime ),
]
```

Week04 basic model-admin

model.py

define a structure of application database

```
from django.db import models
from django.contrib.auth.models import User
# Create your models here.
class Car(models.Model):
	model=models.CharField(max_length=20)
	detail=models.CharField(max_length=100)
	price=models.DecimalField(max_digits=10,decimal_places=2)

class Rent(models.Model):
	user=models.ForeignKey(User, on_delete=models.CASCADE)
	car=models.ForeignKey(Car, on_delete=models.CASCADE)
	start=models.DateTimeField()
	stop=models.DateTimeField()
	fee=models.DecimalField(max_digits=10,decimal_places=2)
```

admin.py

database management tool

```
from django.contrib import admin
from rent.models import Rent, Car

class RentAdmin(admin.ModelAdmin):
	list_display=[f.name for f in Rent._meta.fields]
admin.site.register(Rent,RentAdmin)

class CarAdmin(admin.ModelAdmin):
	list_display=[f.name for f in Car._meta.fields]
admin.site.register(Car,CarAdmin)
```

Week05 Web server core concept

Flask

```
from flask import Flask
from flask import render_template
app = Flask(__name__)

@app.route('/')
def home():
    return render_template('home.html', name='Wasit Limprsert')

# 'ipconfig' to check your pubic ip
# you have to disable firewall or allow incomming connection to the server
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Django

Same concept with a solid structure

views.py

```
from django.shortcuts import render

# Create your views here.
def home(request):
      return render(request, 'home.html', {'name': 'Albert Einstein'})
```

urls.py

```
from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home')
]
```

Work with template

```
from django.http import HttpResponse
from django.views.generic.list
import ListView
from .models import Car


# Create your views here.
def home(request):
       print(HttpResponse('Hello
World'))
       return HttpResponse('Hello
World')


class CarListView(ListView):
   model = Car
   template_name='list.html'
```

Project file structure

```
wasitVision/week05_2: tree -F
.
└── myproject/
    ├── db.sqlite3
    ├── manage.py*
    ├── myproject/
    │   ├── __init__.py
    │   ├── __pycache__/
    │   │   ├── __init__.cpython-35.pyc
    │   │   ├── settings.cpython-35.pyc
    │   │   ├── urls.cpython-35.pyc
    │   │   └── wsgi.cpython-35.pyc
    │   ├── settings.py
    │   ├── urls.py
    │   └── wsgi.py
    └── rent/
        ├── admin.py
        ├── apps.py
        ├── __init__.py
        ├── migrations/
        │   ├── 0001_initial.py
        │   ├── __init__.py
        │   └── __pycache__/
        │       ├── 0001_initial.cpython-35.pyc
        │       └── __init__.cpython-35.pyc
        ├── models.py
        ├── __pycache__/
        │   ├── admin.cpython-35.pyc
        │   ├── __init__.cpython-35.pyc
        │   ├── models.cpython-35.pyc
        │   └── views.cpython-35.pyc
        ├── templates/
        │   ├── list.html
        │   └── login.html
        ├── tests.py
        └── views.py
```

Week06 static files

Server static files

There are many ways to serve static contents. In this tutorial we will try the simplest way that uses the file system in the server to serve the static contents, simply by add the static and media urls in the urls.py. STATIC_URL delivers static images, css, js and many types of static file that never be modified by users. In the opposite way, user able to create update delete the file contents in the server by access the MEDIA_URL. Here is the simple example to server those two type of files.

urls.py

```
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]

from django.conf import settings
from django.conf.urls.static import static
urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

File structure

```
wasitVision/static_demo: tree -F
.
└── myproject/
    ├── db.sqlite3
    ├── manage.py*
    ├── media/
    │   ├── cars/
    │   │   └── Screenshot_from_2018-02-12_12-36-29.png
    │   └── media.gif
    ├── myapp/
    │   ├── admin.py
    │   ├── apps.py
    │   ├── __init__.py
    │   ├── migrations/
    │   │   ├── 0001_initial.py
    │   │   ├── 0002_auto_20180228_0741.py
    │   │   ├── __init__.py
    │   │   └── __pycache__/
    │   │       ├── 0001_initial.cpython-35.pyc
    │   │       ├── 0002_auto_20180228_0741.cpython-35.pyc
    │   │       └── __init__.cpython-35.pyc
    │   ├── models.py
    │   ├── __pycache__/
    │   │   ├── admin.cpython-35.pyc
    │   │   ├── __init__.cpython-35.pyc
    │   │   └── models.cpython-35.pyc
    │   ├── tests.py
    │   └── views.py
    ├── myproject/
    │   ├── __init__.py
    │   ├── __pycache__/
    │   │   ├── __init__.cpython-35.pyc
    │   │   ├── settings.cpython-35.pyc
    │   │   ├── urls.cpython-35.pyc
    │   │   └── wsgi.cpython-35.pyc
    │   ├── settings.py
    │   ├── urls.py
    │   └── wsgi.py
    └── static/
        └── hello.txt

10 directories, 28 files
```

Week07 notebook

To run

python manage.py shell_plus --notebook

Basic ORM

```
python manage.py shell_plus --notebook
```

Create a Class diagram

To create a diagram please read a documents of django-extensions here.

```
python ./manage.py graph_models --pydot -a -g -o others/classdiagram.png
```

Select all cars

In [1]:

```
Car.objects.all()
#ORM: Object Relational mapping
```

Out[1]:

```
<QuerySet [<Car: id: 1, model: Vios, price: 500000>, <Car: id: 2, model: Camry, price: 800000>,
<Car: id: 3, model: Jazz, price: 400000>]>
```

In [2]:

```
print(Car.objects.all().query)
```

```
SELECT "myapp_car"."id", "myapp_car"."model", "myapp_car"."detail", "myapp_car"."price"
FROM "myapp_car"
```

In [3]:

```
for i in Car.objects.all():
    print(i.model, i.detail, i.price)
```

```
Vios medium price 500000
Camry High price 800000
Jazz low price entry car 400000
```

get a car by id

In [4]:

```
Customer.objects.get(id=1)
```

Out[4]:

```
<Customer: id: 1, Albert>
```

get cars by filter

In [5]:

```
Car.objects.filter(price__lte=500000)
```

Out[5]:

```
<QuerySet [<Car: id: 1, model: Vios, price: 500000>, <Car: id: 3, model: Jazz, price: 400000>]>
```

In [6]:

```
Car.objects.filter(price=500000)
```

Out[6]:

```
<QuerySet [<Car: id: 1, model: Vios, price: 500000>]>
```

In [7]:

```
Car.objects.filter(price__gt=500000)
```

Out[7]:

```
<QuerySet [<Car: id: 2, model: Camry, price: 800000>]>
```

In [8]:

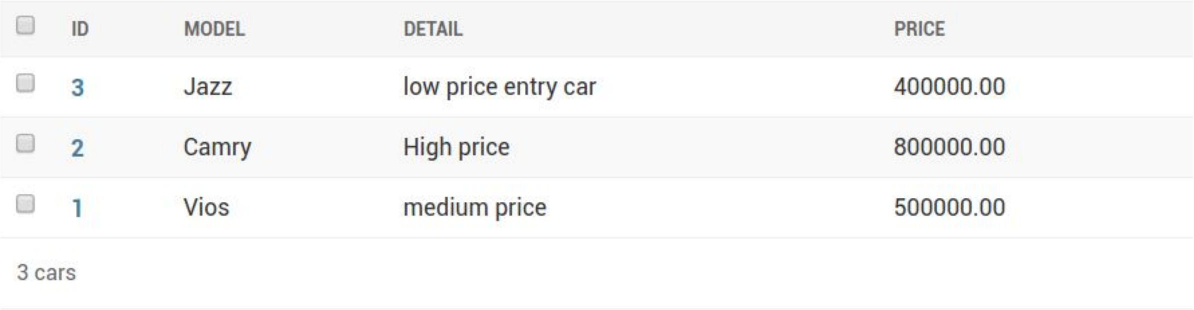
```
print( Car.objects.filter(price__gte=500000).query )
```

```
SELECT "myapp_car"."id", "myapp_car"."model", "myapp_car"."detail", "myapp_car"."price"
FROM "myapp_car" WHERE "myapp_car"."price" >= 500000
```

Relation

ORM can resolve forward and reward relation

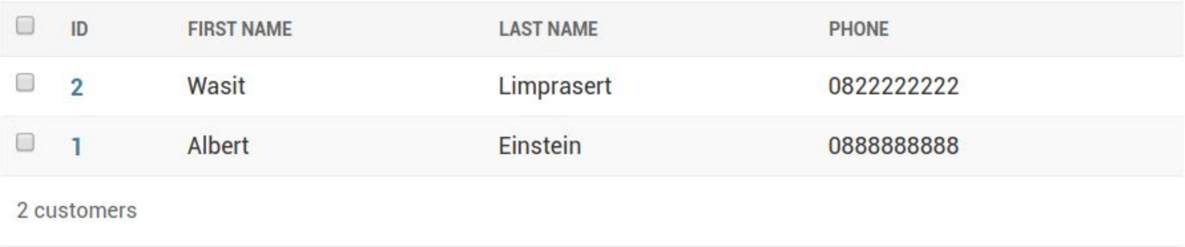
Car



□	ID	MODEL	DETAIL	PRICE
□	3	Jazz	low price entry car	400000.00
□	2	Camry	High price	800000.00
□	1	Vios	medium price	500000.00

3 cars

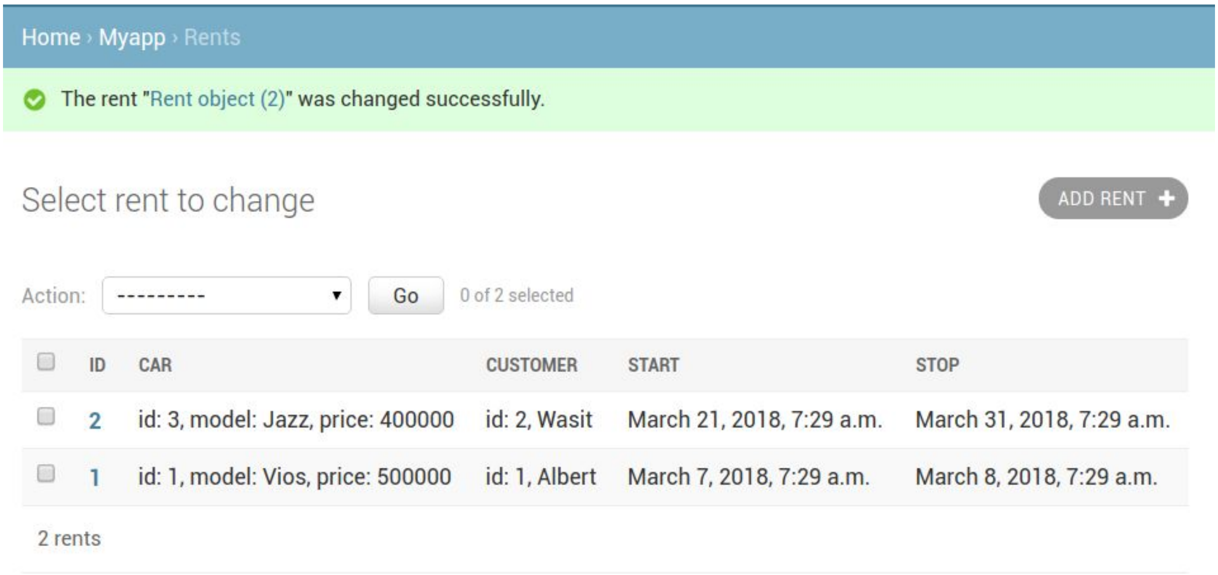
Customer



□	ID	FIRST NAME	LAST NAME	PHONE
□	2	Wasit	Limprasert	0822222222
□	1	Albert	Einstein	0888888888

2 customers

Rent



Home › Myapp › Rents

The rent "Rent object (2)" was changed successfully.

Select rent to change

Action: --------- ▾ Go 0 of 2 selected

□	ID	CAR	CUSTOMER	START	STOP
□	2	id: 3, model: Jazz, price: 400000	id: 2, Wasit	March 21, 2018, 7:29 a.m.	March 31, 2018, 7:29 a.m.
□	1	id: 1, model: Vios, price: 500000	id: 1, Albert	March 7, 2018, 7:29 a.m.	March 8, 2018, 7:29 a.m.

2 rents

```
In [9]:
Rent.objects.get(id=2).car
Out[9]:
<Car: id: 3, model: Jazz, price: 400000>
In [10]:
Rent.objects.get(id=2).customer
Out[10]:
<Customer: id: 2, Wasit>
In [11]:
Rent.objects.filter(car__price__lte=400000)
Out[11]:
<QuerySet [<Rent: Rent object (2)>]>
In [12]:
Rent.objects.filter(car__price__lte=500000)
Out[12]:
<QuerySet [<Rent: Rent object (1)>, <Rent: Rent object (2)>]>
In [13]:
Rent.objects.filter(car__price__lte=500000, customer__first_name='Albert') #AND
conjunction
Out[13]:
<QuerySet [<Rent: Rent object (1)>]>
```

Week09 REST API

setting.py

```
INSTALLED_APPS = [
    ...,
    'rest_framework',
]
```

routers.py

```
from rest_framework import routers, serializers, viewsets
from myapp.viewsets import CustomerViewSet, CarViewSet, RentViewSet
router = routers.DefaultRouter()
router.register(r'customer', CustomerViewSet)
router.register(r'car', CarViewSet)
router.register(r'rent', RentViewSet)
```

serializers.py

```

from myapp.models import Customer, Car, Rent
from rest_framework import routers, serializers, viewsets



class CustomerSerializer(serializers.ModelSerializer):
    class Meta:
        model = Customer
        fields = '__all__'

class CarSerializer(serializers.ModelSerializer):
    class Meta:
        model = Car
        fields = '__all__'

class RentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Rent
        fields = '__all__'
```

viewsets.py

```
from rest_framework import routers, serializers, viewsets
from myapp.models import Customer, Car, Rent
from myapp.serializers import CustomerSerializer, CarSerializer, RentSerializer

class CustomerViewSet(viewsets.ModelViewSet):
    queryset = Customer.objects.all()
    serializer_class = CustomerSerializer

class CarViewSet(viewsets.ModelViewSet):
    queryset = Car.objects.all()
    serializer_class = CarSerializer

class RentViewSet(viewsets.ModelViewSet):
    queryset = Rent.objects.all()
    serializer_class = RentSerializer
```

Week11 Query

01 Import Data

Setting some useful variables

In [1]:

```
import os
import re
APP_NAME="myapp"
ROOT_PATH=os.path.abspath(".")
print "ROOT_PATH: %s"%ROOT_PATH

input_filename=os.path.join(ROOT_PATH,"rent_input.xls")
print input_filename
import datetime
```

```
ROOT_PATH: C:\Users\Wasit\Documents\GitHub\cs459_final\myproject
C:\Users\Wasit\Documents\GitHub\cs459_final\myproject\rent_input.xls
```

Loading Car detail from the 1st sheet from the input excel

In [2]:

```
import pandas as pd
cvt={
        0:int,
        1:unicode,
        2:unicode,
        3:float,
        4:unicode,
        5:int,
    }
df_car=pd.read_excel(io=input_filename,sheetname=0,converters=cvt)
```

In [3]:

```
print df_car
```

```
   ID   CarMaker CarModel  CarPrice   CarColor  CarYear
0   1  Mitsubishi     L200    9995.0        red     2001
```

```
1   2       Mini   Cooper   12500.0       red     2005
2   3        TVR   Tuscan   18000.0      blue     2003
3   4        BMW       Z3   13995.0    silver     2002
4   5     Toyota   Celica    4665.0  dark blue    1997
5   6       Audi       TT   21995.0    silver     2005
6   7   Mercedes     E320   15495.0     green     2004
```

Example for iterate over all row in the sheet

In [4]:

```
for k,i in df_car.iterrows():
    print "k: %s, i:%s\n%s"%(k, i, "-"*30)
```

```
k: 0, i:ID              1
CarMaker    Mitsubishi
CarModel          L200
CarPrice          9995
CarColor           red
CarYear           2001
Name: 0, dtype: object
------------------------------
k: 1, i:ID          2
CarMaker      Mini
CarModel    Cooper
CarPrice     12500
CarColor       red
CarYear       2005
Name: 1, dtype: object
------------------------------
k: 2, i:ID          3
CarMaker       TVR
CarModel    Tuscan
CarPrice     18000
CarColor      blue
CarYear       2003
Name: 2, dtype: object
------------------------------
k: 3, i:ID          4
CarMaker       BMW
CarModel        Z3
CarPrice     13995
CarColor    silver
CarYear       2002
Name: 3, dtype: object
------------------------------
```

```
k: 4, i:ID                5
CarMaker       Toyota
CarModel       Celica
CarPrice         4665
CarColor    dark blue
CarYear          1997
Name: 4, dtype: object
------------------------------
k: 5, i:ID             6
CarMaker       Audi
CarModel         TT
CarPrice      21995
CarColor     silver
CarYear        2005
Name: 5, dtype: object
------------------------------
k: 6, i:ID               7
CarMaker     Mercedes
CarModel         E320
CarPrice        15495
CarColor        green
CarYear          2004
Name: 6, dtype: object
------------------------------
```

Each row contains 6 columns as following

In [5]:

```
i
```

Out[5]:

```
ID                 7
CarMaker     Mercedes
CarModel         E320
CarPrice        15495
CarColor        green
CarYear          2004
Name: 6, dtype: object
```

Uploading the row i into database

In [6]:

```
import pytz
year=datetime.datetime(year=i['CarYear'], month=1, day=1, tzinfo=pytz.UTC)
kargs={
    'maker':i['CarMaker'],
    'price':i['CarPrice'],
    'model':i['CarModel'],
    'year': year
}
car, created = Car.objects.update_or_create(
                    id=i['ID'],
                    defaults=kargs
                )
```

Now uploading every rows

In [7]:

```
for k,i in df_car.iterrows():
    year=datetime.datetime(year=i['CarYear'], month=1,day=1, tzinfo=pytz.UTC)
    kargs={
        'maker':i['CarMaker'],
        'price':i['CarPrice'],
        'model':i['CarModel'],
        'year': year
    }
    car, created = Car.objects.update_or_create(
                        id=i['ID'],
                        defaults=kargs
                    )
```

Then loading customer from the 2nd sheet

In [8]:

```
import pandas as pd
cvt={
        0:unicode,
        1:unicode,
        2:unicode,
        3:unicode,
        4:unicode,
        5:unicode,
        6:unicode
    }
df_customer=pd.read_excel(io=input_filename,sheetname=1,converters=cvt)
keys=df_customer.keys()
print keys
```

```
Index([u'ID', u'ClientFirstName', u'ClientLastName', u'ClientAddress',
       u'Postcode', u'Tel', u'Email'],
      dtype='object')
```

And uploading the customer

In [9]:

```
for k,i in df_customer.iterrows():
    kargs={
        'first_name':i[keys[1]],
        'last_name':i[keys[2]],
        'Address':i[keys[3]],
        'postcode':i[keys[4]],
        'telephone':i[keys[5]],
        'email':i[keys[6]]
    }
    customer, created = Customer.objects.update_or_create(
                        id=i['ID'],
                        defaults=kargs
                    )
```

Finally Loading the 3rd sheet

In [10]:

```
import pandas as pd
df_rent=pd.read_excel(io=input_filename,sheetname=2)
keys=df_rent.keys()
print keys
```

```
Index([u'ID', u'RentDate', u'ServiceCost', u'ReturnDate', u'ClientID',
       u'CarID'],
      dtype='object')
```

In [11]:

```
df_rent
```

Out[11]:

	ID	RentDate	ServiceCost	ReturnDate	ClientID	CarID
0	1	2014-03-12	549.75	2014-03-17	1	1
1	2	2014-03-12	1050.00	2014-03-20	2	2
2	3	2014-03-13	1310.00	2014-03-20	3	3
3	4	2014-03-17	425.00	2014-03-20	1	2

4	5	2014-03-20	189.95	2014-03-21	4	4
5	6	2014-03-20	50.00	2014-03-20	2	5
6	7	2014-03-20	269.95	2014-03-21	2	6
7	8	2014-03-21	514.85	2014-03-24	5	7
8	9	2014-03-24	549.75	2014-03-29	6	1
9	10	2014-03-29	50.00	2014-03-29	1	2
10	11	2014-03-29	500.00	2014-03-29	5	3
11	12	2014-03-29	500.00	2014-03-29	7	4
12	13	2014-03-30	430.00	2014-03-29	2	1
13	14	2014-03-30	430.00	2014-03-29	6	3
14	15	2014-03-30	430.00	2014-03-29	1	4
15	16	2014-03-30	430.00	2014-03-29	5	5
16	17	2014-03-30	430.00	2014-03-29	6	6

And uploading Rent records to the database

In [12]:

```
for k,i in df_rent.iterrows():
    utc=pytz.timezone('UTC')
    kargs={
        'rent_date': utc.localize(i['RentDate']),
        'return_date':utc.localize(i['ReturnDate']),
        'cost':i['ServiceCost'],
        'car': Car.objects.get(id=i['CarID']),
        'customer': Customer.objects.get(id=i['ClientID']),
    }
    customer, created = Rent.objects.update_or_create(
                        id=i['ID'],
                        defaults=kargs
                    )
```

Code explain

To convert from naive-datetime to time-zone-aware-datetime

```
import pytz
utc=pytz.timezone('UTC')
utc.localize( your_datetime )
```

02 Query

Query Pattern

- What is total rental cost between 13/03/2014-24/03/2014?
- How much money collected from the car id=2?

Getting a record by id

In [2]:

```
c=Customer.objects.get(id=2)
print(c)
```

```
Customer object (2)
```

Getting all records from table Customer

In [3]:

```
Customer.objects.all()
```

Out[3]:

```
<QuerySet [<Customer: Customer object (1)>, <Customer: Customer object (2)>, <Customer: Customer object (3)>, <Customer: Customer object (4)>, <Customer: Customer object (5)>, <Customer: Customer object (6)>, <Customer: Customer object (7)>, <Customer: Customer object (8)>, <Customer: Customer object (9)>, <Customer: Customer object (10)>]>
```

In [3]:

```
# SQL command
print Customer.objects.all().query
```

```
SELECT "myapp_customer"."id", "myapp_customer"."first_name", "myapp_customer"."last_name", "myapp_customer"."Address", "myapp_customer"."postcode", "myapp_customer"."telephone", "myapp_customer"."email" FROM "myapp_customer"
```

Filter records within range

In [4]:

```
from datetime import datetime
import pytz
utc=pytz.timezone('UTC')
start_date = utc.localize( datetime.strptime('2014-03-13','%Y-%m-%d') )
stop_date = utc.localize( datetime.strptime('2014-03-24','%Y-%m-%d') )
```

In [5]:

```
Rent.objects.filter(rent_date__range=[start_date, stop_date])
```

Out[5]:

```
[<Rent: id: 3>, <Rent: id: 4>, <Rent: id: 5>, <Rent: id: 6>, <Rent: id: 7>, <Rent: id: 8>, <Rent: id: 9>]
```

In [6]:

```
# SQL command
print Rent.objects.filter(rent_date__range=[start_date, stop_date ]).query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM "myapp_rent"
WHERE "myapp_rent"."rent_date" BETWEEN 2014-03-13 00:00:00 AND 2014-03-24 00:00:00
```

Filter less_than_or_equal (__lte)

In [7]:

```
# rent that happended before or equal 13 March 2014
Rent.objects.filter(rent_date__lte=start_date)
```

Out[7]:

```
[<Rent: id: 1>, <Rent: id: 2>, <Rent: id: 3>]
```

In [8]:

```
# SQL command
print Rent.objects.filter(rent_date__lte=start_date).query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM "myapp_rent"
WHERE "myapp_rent"."rent_date" <= 2014-03-13 00:00:00
```

Filter greater than (__gt)

In [9]:

```
# rent that happended after 13 March 2014
Rent.objects.filter(rent_date__gt=start_date)
```

Out[9]:

```
[<Rent: id: 4>, <Rent: id: 5>, <Rent: id: 6>, <Rent: id: 7>, <Rent: id: 8>, <Rent: id: 9>, <Rent: id: 10>,
<Rent: id: 11>, <Rent: id: 12>, <Rent: id: 13>, <Rent: id: 14>, <Rent: id: 15>, <Rent: id: 16>, <Rent: id:
17>]
```

In [10]:

```
# SQL command
print Rent.objects.filter(rent_date__gt=start_date).query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM "myapp_rent"
WHERE "myapp_rent"."rent_date" > 2014-03-13 00:00:00
```

What is total rental cost between 13/03/2014-24/03/2014?

Naive solution (but slow)

In [11]:

```
%%timeit -n10
total=0
q=Rent.objects.filter(rent_date__range=[start_date, stop_date])
for i in q:
   total=total + i.cost
```

```
10 loops, best of 3: 2.33 ms per loop
```

Better by Using "aggregration()"

In [12]:

```
%%timeit -n10
from django.db.models import Sum, Max, Min, Avg
Rent.objects.filter(rent_date__range=[start_date,
stop_date]).aggregate(Sum('cost'))
```

```
10 loops, best of 3: 879 µs per loop
```

In [13]:

```
q=Rent.objects.filter(rent_date__range=[start_date, stop_date])
r=q.aggregate(Sum('cost'))
r
```

Out[13]:

```
{'cost__sum': Decimal('3309.50')}
```

In [14]:

```
Rent.objects.filter(rent_date__range=[start_date, stop_date]).aggregate(Max('cost'))
```

Out[14]:

```
{'cost__max': Decimal('1310.00')}
```

Annotate Count

In [15]:

```
from django.db.models import Count
```

In [16]:

```
q=Car.objects.annotate(Count("rent"))
```

In [17]:

```
q[0].rent__count
```

Out[17]:

```
3
```

In [18]:

```
for i in q:
    print "rent__count:%s car:%s"%(i.rent__count, i)
```

```
rent__count:3 car:id: 1, Mitsubishi L200
rent__count:3 car:id: 2, Mini Cooper
rent__count:3 car:id: 3, TVR Tuscan
rent__count:3 car:id: 4, BMW Z3
rent__count:2 car:id: 5, Toyota Celica
rent__count:2 car:id: 6, Audi TT
rent__count:1 car:id: 7, Mercedes E320
```

In [19]:

```
print Car.objects.annotate(Count("rent")).query
```

```
SELECT "myapp_car"."id", "myapp_car"."maker", "myapp_car"."price",
"myapp_car"."model", "myapp_car"."year", COUNT("myapp_rent"."id") AS
"rent__count" FROM "myapp_car" LEFT OUTER JOIN "myapp_rent" ON
("myapp_car"."id" = "myapp_rent"."car_id") GROUP BY "myapp_car"."id",
"myapp_car"."maker", "myapp_car"."price", "myapp_car"."model",
"myapp_car"."year"
```

Reverse relation

In [20]:

```
Car.objects.get(id=2)
```

Out[20]:

```
<Car: id: 2, Mini Cooper>
```

In [21]:

```
Car.objects.get(id=2).rent_set.all()
```

Out[21]:

```
[<Rent: id: 2>, <Rent: id: 4>, <Rent: id: 10>]
```

In [22]:

```
# SQL command
print Car.objects.get(id=2).rent_set.all().query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."car_id" = 2
```

How much money collected from the car id=2?

Reverse relation (slow)

In [23]:

```
%%timeit -n1
sum_cost=Car.objects.get(id=2).rent_set.all().aggregate(Sum('cost'))
print sum_cost
```

```
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
1 loop, best of 3: 2.03 ms per loop
```

In [24]:

```
print Car.objects.get(id=2).rent_set.all().query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."car_id" = 2
```

Forward relation

In [25]:

```
%%timeit -n1
sum_cost=Rent.objects.filter(car__id=2).aggregate(Sum('cost'))
print sum_cost
```

```
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
1 loop, best of 3: 2.27 ms per loop
```

In [26]:

```
print Rent.objects.filter(car__id=2).query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."car_id" = 2
```

Find total income for each car

In [27]:

```
q=Car.objects.annotate(Sum("rent__cost"))
for i in q:
    print "income:%s car:%s"%(i.rent__cost__sum,i)
```

```
income:1529.50 car:id: 1, Mitsubishi L200
income:1525.00 car:id: 2, Mini Cooper
income:2240.00 car:id: 3, TVR Tuscan
income:1119.95 car:id: 4, BMW Z3
income:480.00 car:id: 5, Toyota Celica
income:699.95 car:id: 6, Audi TT
income:514.85 car:id: 7, Mercedes E320
```

Q: Why do we need to use revese relation?

A: Sometimes we need to iterate over all cars to get total cost of each car.

In [28]:

```
%%timeit -n1
for i in Car.objects.all():
    print "%s\n    %s"%( i, i.rent_set.all().aggregate(Sum('cost')) )
```

```
id: 1, Mitsubishi L200
    {'cost__sum': Decimal('1529.50')}
id: 2, Mini Cooper
    {'cost__sum': Decimal('1525.00')}
id: 3, TVR Tuscan
    {'cost__sum': Decimal('2240.00')}
id: 4, BMW Z3
    {'cost__sum': Decimal('1119.95')}
id: 5, Toyota Celica
    {'cost__sum': Decimal('480.00')}
id: 6, Audi TT
    {'cost__sum': Decimal('699.95')}
id: 7, Mercedes E320
    {'cost__sum': Decimal('514.85')}
id: 1, Mitsubishi L200
    {'cost__sum': Decimal('1529.50')}
id: 2, Mini Cooper
    {'cost__sum': Decimal('1525.00')}
id: 3, TVR Tuscan
    {'cost__sum': Decimal('2240.00')}
id: 4, BMW Z3
    {'cost__sum': Decimal('1119.95')}
id: 5, Toyota Celica
    {'cost__sum': Decimal('480.00')}
id: 6, Audi TT
    {'cost__sum': Decimal('699.95')}
id: 7, Mercedes E320
    {'cost__sum': Decimal('514.85')}
```

```
id: 1, Mitsubishi L200
    {'cost__sum': Decimal('1529.50')}
id: 2, Mini Cooper
    {'cost__sum': Decimal('1525.00')}
id: 3, TVR Tuscan
    {'cost__sum': Decimal('2240.00')}
id: 4, BMW Z3
    {'cost__sum': Decimal('1119.95')}
id: 5, Toyota Celica
    {'cost__sum': Decimal('480.00')}
id: 6, Audi TT
    {'cost__sum': Decimal('699.95')}
id: 7, Mercedes E320
    {'cost__sum': Decimal('514.85')}
1 loop, best of 3: 8.94 ms per loop
```

Better Solution by using "annotation()"

In [29]:

```
%%timeit -n1
cars=Car.objects.all().annotate(Sum('rent__cost'))
for i in cars:
    print "%s\n    %s"%( i, i.rent__cost__sum )
```

```
id: 1, Mitsubishi L200
    1529.50
id: 2, Mini Cooper
    1525.00
id: 3, TVR Tuscan
    2240.00
id: 4, BMW Z3
    1119.95
id: 5, Toyota Celica
    480.00
id: 6, Audi TT
    699.95
id: 7, Mercedes E320
    514.85
id: 1, Mitsubishi L200
    1529.50
```

```
id: 2, Mini Cooper
   1525.00
id: 3, TVR Tuscan
   2240.00
id: 4, BMW Z3
   1119.95
id: 5, Toyota Celica
   480.00
id: 6, Audi TT
   699.95
id: 7, Mercedes E320
   514.85
id: 1, Mitsubishi L200
   1529.50
id: 2, Mini Cooper
   1525.00
id: 3, TVR Tuscan
   2240.00
id: 4, BMW Z3
   1119.95
id: 5, Toyota Celica
   480.00
id: 6, Audi TT
   699.95
id: 7, Mercedes E320
   514.85
1 loop, best of 3: 6.45 ms per loop
```

In [30]:

```
print Car.objects.all().annotate(Sum('rent__cost')).query
```

```
SELECT "myapp_car"."id", "myapp_car"."maker", "myapp_car"."price",
"myapp_car"."model", "myapp_car"."year", CAST(SUM("myapp_rent"."cost") AS
NUMERIC) AS "rent__cost__sum" FROM "myapp_car" LEFT OUTER JOIN
"myapp_rent" ON ("myapp_car"."id" = "myapp_rent"."car_id") GROUP BY
"myapp_car"."id", "myapp_car"."maker", "myapp_car"."price", "myapp_car"."model",
"myapp_car"."year"
```

03 complex query

Query Pattern

- What is total rental cost between 13/03/2014-24/03/2014?
- How much money collected from the car id=2?

Getting a record by id

In [2]:

```
c=Customer.objects.get(id=2)
print(c)
```

```
Customer object (2)
```

Getting all records from table Customer

In [3]:

```
Customer.objects.all()
```

Out[3]:

```
<QuerySet [<Customer: Customer object (1)>, <Customer: Customer object (2)>, <Customer: Customer object (3)>, <Customer: Customer object (4)>, <Customer: Customer object (5)>, <Customer: Customer object (6)>, <Customer: Customer object (7)>, <Customer: Customer object (8)>, <Customer: Customer object (9)>, <Customer: Customer object (10)>]>
```

In [3]:

```
# SQL command
print Customer.objects.all().query
```

```
SELECT "myapp_customer"."id", "myapp_customer"."first_name", "myapp_customer"."last_name", "myapp_customer"."Address", "myapp_customer"."postcode", "myapp_customer"."telephone", "myapp_customer"."email" FROM "myapp_customer"
```

Filter records within range

In [4]:

```
from datetime import datetime
import pytz
utc=pytz.timezone('UTC')
start_date = utc.localize( datetime.strptime('2014-03-13','%Y-%m-%d') )
stop_date = utc.localize( datetime.strptime('2014-03-24','%Y-%m-%d') )
```

In [5]:

```
Rent.objects.filter(rent_date__range=[start_date, stop_date])
```

Out[5]:

```
[<Rent: id: 3>, <Rent: id: 4>, <Rent: id: 5>, <Rent: id: 6>, <Rent: id: 7>, <Rent: id: 8>,
<Rent: id: 9>]
```

In [6]:

```
# SQL command
print Rent.objects.filter(rent_date__range=[start_date, stop_date ]).query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."rent_date" BETWEEN 2014-03-13 00:00:00
AND 2014-03-24 00:00:00
```

Filter less_than_or_equal (__lte)

In [7]:

```
# rent that happended before or equal 13 March 2014
Rent.objects.filter(rent_date__lte=start_date)
```

Out[7]:

```
[<Rent: id: 1>, <Rent: id: 2>, <Rent: id: 3>]
```

In [8]:

```
# SQL command
print Rent.objects.filter(rent_date__lte=start_date).query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date", "myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM "myapp_rent" WHERE "myapp_rent"."rent_date" <= 2014-03-13 00:00:00
```

Filter greater than (__gt)

In [9]:

```
# rent that happended after 13 March 2014
Rent.objects.filter(rent_date__gt=start_date)
```

Out[9]:

```
[<Rent: id: 4>, <Rent: id: 5>, <Rent: id: 6>, <Rent: id: 7>, <Rent: id: 8>, <Rent: id: 9>, <Rent: id: 10>, <Rent: id: 11>, <Rent: id: 12>, <Rent: id: 13>, <Rent: id: 14>, <Rent: id: 15>, <Rent: id: 16>, <Rent: id: 17>]
```

In [10]:

```
# SQL command
print Rent.objects.filter(rent_date__gt=start_date).query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date", "myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM "myapp_rent" WHERE "myapp_rent"."rent_date" > 2014-03-13 00:00:00
```

What is total rental cost between 13/03/2014-24/03/2014?

Naive solution (but slow)

In [11]:

```
%%timeit -n10
total=0
q=Rent.objects.filter(rent_date__range=[start_date, stop_date])
for i in q:
   total=total + i.cost
```

```
10 loops, best of 3: 2.33 ms per loop
```

Better by Using "aggregration()"

In [12]:

```
%%timeit -n10
from django.db.models import Sum, Max, Min, Avg
Rent.objects.filter(rent_date__range=[start_date,
stop_date]).aggregate(Sum('cost'))
```

```
10 loops, best of 3: 879 µs per loop
```

In [13]:

```
q=Rent.objects.filter(rent_date__range=[start_date, stop_date])
r=q.aggregate(Sum('cost'))
r
```

Out[13]:

```
{'cost__sum': Decimal('3309.50')}
```

In [14]:

```
Rent.objects.filter(rent_date__range=[start_date, stop_date]).aggregate(Max('cost'))
```

Out[14]:

```
{'cost__max': Decimal('1310.00')}
```

Annotate Count

In [15]:

```
from django.db.models import Count
```

In [16]:

```
q=Car.objects.annotate(Count("rent"))
```

In [17]:

```
q[0].rent__count
```

Out[17]:

```
3
```

In [18]:

```
for i in q:
    print "rent__count:%s car:%s"%(i.rent__count, i)
```

```
rent__count:3 car:id: 1, Mitsubishi L200
rent__count:3 car:id: 2, Mini Cooper
rent__count:3 car:id: 3, TVR Tuscan
rent__count:3 car:id: 4, BMW Z3
rent__count:2 car:id: 5, Toyota Celica
rent__count:2 car:id: 6, Audi TT
rent__count:1 car:id: 7, Mercedes E320
```

In [19]:

```python
print Car.objects.annotate(Count("rent")).query
```

```
SELECT "myapp_car"."id", "myapp_car"."maker", "myapp_car"."price",
"myapp_car"."model", "myapp_car"."year", COUNT("myapp_rent"."id") AS
"rent__count" FROM "myapp_car" LEFT OUTER JOIN "myapp_rent" ON
("myapp_car"."id" = "myapp_rent"."car_id") GROUP BY "myapp_car"."id",
"myapp_car"."maker", "myapp_car"."price", "myapp_car"."model",
"myapp_car"."year"
```

Reverse relation

In [20]:

```python
Car.objects.get(id=2)
```

Out[20]:

```
<Car: id: 2, Mini Cooper>
```

In [21]:

```python
Car.objects.get(id=2).rent_set.all()
```

Out[21]:

```
[<Rent: id: 2>, <Rent: id: 4>, <Rent: id: 10>]
```

In [22]:

```python
# SQL command
print Car.objects.get(id=2).rent_set.all().query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."car_id" = 2
```

How much money collected from the car id=2?

Reverse relation (slow)

In [23]:

```
%%timeit -n1
sum_cost=Car.objects.get(id=2).rent_set.all().aggregate(Sum('cost'))
print sum_cost
```

```
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
1 loop, best of 3: 2.03 ms per loop
```

In [24]:

```
print Car.objects.get(id=2).rent_set.all().query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
"myapp_rent" WHERE "myapp_rent"."car_id" = 2
```

Forward relation

In [25]:

```
%%timeit -n1
sum_cost=Rent.objects.filter(car__id=2).aggregate(Sum('cost'))
print sum_cost
```

```
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
{'cost__sum': Decimal('1525.00')}
1 loop, best of 3: 2.27 ms per loop
```

In [26]:

```
print Rent.objects.filter(car__id=2).query
```

```
SELECT "myapp_rent"."id", "myapp_rent"."rent_date", "myapp_rent"."return_date",
"myapp_rent"."cost", "myapp_rent"."car_id", "myapp_rent"."customer_id" FROM
```

```
"myapp_rent" WHERE "myapp_rent"."car_id" = 2
```

Find total income for each car

In [27]:

```
q=Car.objects.annotate(Sum("rent__cost"))
for i in q:
    print "income:%s car:%s"%(i.rent__cost__sum,i)
```

```
income:1529.50 car:id: 1, Mitsubishi L200
income:1525.00 car:id: 2, Mini Cooper
income:2240.00 car:id: 3, TVR Tuscan
income:1119.95 car:id: 4, BMW Z3
income:480.00 car:id: 5, Toyota Celica
income:699.95 car:id: 6, Audi TT
income:514.85 car:id: 7, Mercedes E320
```

Q: Why do we need to use reverse relation?

A: Sometimes we need to iterate over all cars to get total cost of each car.

In [28]:

```
%%timeit -n1
for i in Car.objects.all():
    print "%s\n    %s"%( i, i.rent_set.all().aggregate(Sum('cost')) )
```

```
id: 1, Mitsubishi L200
    {'cost__sum': Decimal('1529.50')}
id: 2, Mini Cooper
    {'cost__sum': Decimal('1525.00')}
id: 3, TVR Tuscan
    {'cost__sum': Decimal('2240.00')}
id: 4, BMW Z3
    {'cost__sum': Decimal('1119.95')}
id: 5, Toyota Celica
    {'cost__sum': Decimal('480.00')}
id: 6, Audi TT
```

```
   {'cost__sum': Decimal('699.95')}
id: 7, Mercedes E320
   {'cost__sum': Decimal('514.85')}
id: 1, Mitsubishi L200
   {'cost__sum': Decimal('1529.50')}
id: 2, Mini Cooper
   {'cost__sum': Decimal('1525.00')}
id: 3, TVR Tuscan
   {'cost__sum': Decimal('2240.00')}
id: 4, BMW Z3
   {'cost__sum': Decimal('1119.95')}
id: 5, Toyota Celica
   {'cost__sum': Decimal('480.00')}
id: 6, Audi TT
   {'cost__sum': Decimal('699.95')}
id: 7, Mercedes E320
   {'cost__sum': Decimal('514.85')}
id: 1, Mitsubishi L200
   {'cost__sum': Decimal('1529.50')}
id: 2, Mini Cooper
   {'cost__sum': Decimal('1525.00')}
id: 3, TVR Tuscan
   {'cost__sum': Decimal('2240.00')}
id: 4, BMW Z3
   {'cost__sum': Decimal('1119.95')}
id: 5, Toyota Celica
   {'cost__sum': Decimal('480.00')}
id: 6, Audi TT
   {'cost__sum': Decimal('699.95')}
id: 7, Mercedes E320
   {'cost__sum': Decimal('514.85')}
1 loop, best of 3: 8.94 ms per loop
```

Better Solution by using "annotation()"

In [29]:

```
%%timeit -n1
cars=Car.objects.all().annotate(Sum('rent__cost'))
for i in cars:
    print "%s\n    %s"%( i, i.rent__cost__sum )
```

```
id: 1, Mitsubishi L200
    1529.50
id: 2, Mini Cooper
    1525.00
id: 3, TVR Tuscan
    2240.00
id: 4, BMW Z3
    1119.95
id: 5, Toyota Celica
    480.00
id: 6, Audi TT
    699.95
id: 7, Mercedes E320
    514.85
id: 1, Mitsubishi L200
    1529.50
id: 2, Mini Cooper
    1525.00
id: 3, TVR Tuscan
    2240.00
id: 4, BMW Z3
    1119.95
id: 5, Toyota Celica
    480.00
id: 6, Audi TT
    699.95
id: 7, Mercedes E320
    514.85
id: 1, Mitsubishi L200
    1529.50
id: 2, Mini Cooper
    1525.00
id: 3, TVR Tuscan
    2240.00
id: 4, BMW Z3
    1119.95
id: 5, Toyota Celica
    480.00
```

```
id: 6, Audi TT
    699.95
id: 7, Mercedes E320
    514.85
1 loop, best of 3: 6.45 ms per loop
```

In [30]:

```
print Car.objects.all().annotate(Sum('rent__cost')).query
```

```
SELECT "myapp_car"."id", "myapp_car"."maker", "myapp_car"."price",
"myapp_car"."model", "myapp_car"."year", CAST(SUM("myapp_rent"."cost") AS
NUMERIC) AS "rent__cost__sum" FROM "myapp_car" LEFT OUTER JOIN
"myapp_rent" ON ("myapp_car"."id" = "myapp_rent"."car_id") GROUP BY
"myapp_car"."id", "myapp_car"."maker", "myapp_car"."price", "myapp_car"."model",
"myapp_car"."year"
```

In [ ]:

Week13 Authentication

views.py basic authentication

```
from django.shortcuts import render
from django.http import HttpResponse
from django.contrib.auth import authenticate, login, logout,
update_session_auth_hash
from django.shortcuts import redirect
from django.contrib.auth.decorators import login_required
from django.conf import settings
import sys
from django.contrib.auth.forms import PasswordChangeForm
# Create your views here.
def signin(request):
        if request.method == 'POST' and 'username' in request.POST:
                username = request.POST['username']
                password = request.POST['password']
                user = authenticate(username=username, password=password)
                #print >>sys.stderr, "debug"
                if user is not None:
                        if user.is_active:

                                if 'remember' in request.POST:
                                        #print>>sys.stderr, "%s type:
%s"%(request.POST['remember'],type(request.POST['remember']))
                                        if request.POST['remember']=='1':
                                                request.session.set_expiry(604800)
#remember keep session for a week
                                else:
                                        request.session.set_expiry(14400) #not remember
keep session for 4hrs
                                #print >>sys.stderr, "session expiry:
%s"%request.session.get_expiry_age()

                                login(request, user)
                if 'username' in request.session:
                    pass
                    #print >>sys.stderr, "username_i:
%s"%request.session['username']
                request.session['username'] = user.username
                #print >>sys.stderr, "username_f:
%s"%request.session['username']

                return redirect('http://localhost:8000/admin/')
            else:
                msg="Disabled account"
        else:
            msg="Invalid username or password"
        return render(request,'login.html',{'msg': msg})
    return render(request,'login.html',{'msg': ""})

def signout(request):
    print("signout")
    if 'username' in request.session:
        del request.session['username']
        #print "del uname"
    logout(request)
    return redirect('wl_auth:signin')

@login_required(login_url='wl_auth:signin')
def change_password(request):
    form = PasswordChangeForm(user=request.user)
    #print >>sys.stderr, "request.user: %s"%request.user
    if request.method == 'POST':
        form = PasswordChangeForm(user=request.user, data=request.POST)
        if form.is_valid():
            form.save()
            update_session_auth_hash(request, form.user)
            return redirect('main:home')

    return render(request, 'change_password.html', {
        'form': form,
    })
```

Week14 Deployment with Docker

Please watch a video tutorial before the class
https://www.youtube.com/watch?v=gQe2txpV4eA

docker-compose.yml

```
version: '3.5'
services:
  db:
    container_name: postgres_testcompose
    build: ./postgres
    restart: always

  web1:
    container_name: web1_testcompose
    build: ./myproject
    command: sh /code/run.sh
    ports:
      - 8000:8000
    volumes:
      - ./myproject:/code
    depends_on:
      - db
```

/postgres/Dockerfile

```
FROM postgres
```

/myproject/Dockerfile

```
FROM python:3
ENV PYTHONUNBUFFERED 1
ADD . /code
WORKDIR /code
RUN pip3 install -r requirements.txt
```

File structure

```
wasitVision/week_14: tree -d
.
├── deployment
│   ├── myproject
│   │   ├── myapp
│   │   │   ├── migrations
│   │   │   │   └── __pycache__
│   │   │   └── __pycache__
│   │   ├── myproject
│   │   │   └── __pycache__
│   │   └── www
│   │       ├── media
│   │       │   └── cars
│   │       └── static
│   │           └── admin
│   │               ├── css
│   │               │   └── vendor
│   │               │       └── select2
│   │               ├── fonts
│   │               ├── img
│   │               │   └── gis
│   │               └── js
│   │                   ├── admin
│   │                   └── vendor
│   │                       ├── jquery
│   │                       ├── select2
│   │                       │   └── i18n
│   │                       └── xregexp
│   ├── nginx
│   └── postgres
├── docker_compose
│   ├── myproject
│   │   └── myproject
│   │       └── __pycache__
│   └── postgres
└── docker_file

34 directories
```