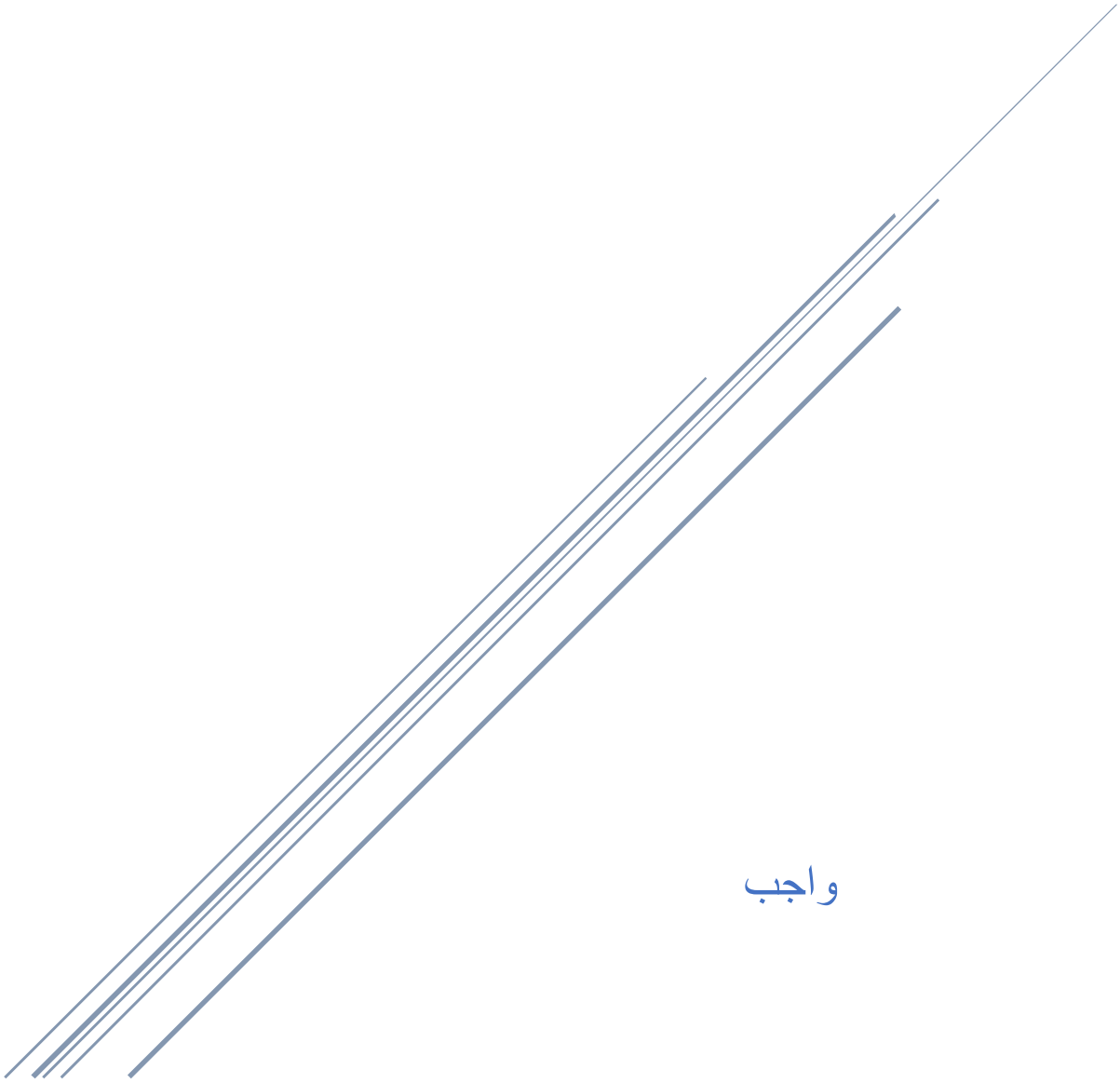
بسم الله الرحمن الرحيم



واجب

اسم الطالب : ريان امين سيف عبد القوي
اسم المقرر :هندسة برمجيات
اسم الأستاذ : مالك المصنف
ايميل : riancomp.ye@gmail.com

بحث شامل عن آلية عمل QuerySet وآلية Clean Up في Django

جدول المحتويات

مقدمة

يعد Django أحد أطر العمل الأكثر شعبية لتطوير تطبيقات الويب باستخدام لغة .Python يوفر Djangoنظام (Object-Relational Mapping) ORM قويًا يسمح للمطورين بالتفاعل مع قواعد البيانات باستخدام كائنات Python بدلاً من كتابة استعلامات SQL مباشرة. يعتبر QuerySet أحد المكونات الأساسية في ORM الخاص بـDjango ، حيث يمثل مجموعة من الكائنات من قاعدة البيانات ويمكن تعديله عبر سلسلة من العمليات.

تعد إدارة البيانات غير المستخدمة والموارد المؤقتة تحديًا مهمًا في تطبيقات الويب. تأتي آلية Clean Upفي Django لمعالجة هذه المشكلة من خلال توفير طرق مختلفة لتنظيف البيانات القديمة تلقائيًا.

QuerySetفيDjango

ما هوQuerySet ؟

QuerySetهو واجهة برمجية تمثل مجموعة من السجلات من قاعدة البيانات. يمكن تعديل QuerySetعبر سلسلة من العمليات مثل التصفية والترتيب والتجميع دون تنفيذ استعلام قاعدة البيانات حتى يتم تقييم النتيجة.

python
```
# مثال أساسي لـ QuerySet
from myapp.models import Product

# إنشاء QuerySet أساسي
all_products = Product.objects.all()

# QuerySet لم يتم تنفيذه بعد
print(all_products)  # <QuerySet []> - لم ينفذ الاستعلام بعد

# تنفيذ QuerySet
products_list = list(all_products)  # الآن ينفذ الاستعلام
```

إنشاء الاستعلامات الأساسية

1. التصفية (Filter)

python
```
# تصفية المنتجات بالسعر
expensive_products = Product.objects.filter(price__gt=100)
available_products = Product.objects.filter(is_available=True)

# تصفية بتعدد الشروط
from django.db.models import Q
products = Product.objects.filter(Q(price__gt=100) | Q(category='electronics'))
```

2. الاستبعاد (Exclude)

python
```
# استبعاد المنتجات غير المتاحة
available_only = Product.objects.exclude(is_available=False)

# استبعاد بشرط مركب
```

```
products = Product.objects.exclude(price__lt=50, category='books')
```

3.التحديد المسبق للعلاقات(Select Related)

python
```
# حسين الاستعلامات للعلاقات (ForeignKey (One-to-One
from myapp.models import Order, Customer

# بدون select_related - يؤدي إلى استعلامات متعددة
orders = Order.objects.all()
for order in orders:
    print(order.customer.name)  # استعلام منفصل لكل طلب

# مع select_related - استعلام واحد مع JOIN
orders = Order.objects.select_related('customer').all()
for order in orders:
    print(order.customer.name)  # لا يوجد استعلامات إضافية
```

4.الجلب المسبق(Prefetch Related)

python
```
# حسين الاستعلامات للعلاقات Many-to-Many و Reverse ForeignKey
from myapp.models import Category, Product

# بدون prefetch_related
categories = Category.objects.all()
for category in categories:
    print([product.name for product in category.products.all()])  # استعلام منفصل لكل فئة

# مع prefetch_related
categories = Category.objects.prefetch_related('products').all()
for category in categories:
    print([product.name for product in category.products.all()])  # استعلامين فقط
```

الاستعلامات المتقدمة

1.التجميع(Aggregate)

python
```
from django.db.models import Avg, Max, Min, Sum
```

```
#حساب إحصائيات المنتجات
stats = Product.objects.aggregate(
    avg_price=Avg('price'),
    max_price=Max('price'),
    total_products=Count('id')
)
#النتيجة: {'avg_price': 150.5, 'max_price': 500, 'total_products': 100}
```

2.التعليق التوضيحي(Annotate)

python
```
from django.db.models import Count, F

#إضافة حقل محسوب لكل منتج
products_with_discount = Product.objects.annotate(
    discounted_price=F('price') * 0.9
)

#حساب عدد المنتجات في كل فئة
categories = Category.objects.annotate(
    product_count=Count('products')
)
```

تحسين الأداء

تقنيات تحسينQuerySet

1. **التقييم الكسول** :**(Lazy Evaluation)** لا ينفذ QuerySet الاستعلام حتى يتم طلب النتيجة.
2. **التخزين المؤقت** :**(Caching)** عند تقييمQuerySet ، يتم تخزين النتيجة مؤقتًا.
3. **التحديد الانتقائي للحقول**:**(only/defer)**

python
```
#جلب الحقول المحددة فقط
products = Product.objects.only('name', 'price')

#تأجيل جلب الحقول الكبيرة
products = Product.objects.defer('description')
```

آلية Clean Up فيDjango

أهمية التنظيف في تطبيقات الويب

- تحسين أداء التطبيق
- توفير مساحة التخزين
- الحفاظ على خصوصية البيانات
- الامتثال للوائح حماية البيانات

الأساليب المختلفة للتنظيف

1أوامر الإدارة(Management Commands)

python
```python
# myapp/management/commands/cleanup_sessions.py
from django.core.management.base import BaseCommand
from django.contrib.sessions.models import Session
from django.utils import timezone

class Command(BaseCommand):
    help = 'Clean up expired sessions'

    def handle(self, *args, **options):
        expired_sessions = Session.objects.filter(expire_date__lt=timezone.no
w())
        count = expired_sessions.count()
        expired_sessions.delete()
        self.stdout.write(f'Deleted {count} expired sessions')
```

.2الإشارات(Signals)

python
```python
# models.py
from django.db import models
from django.db.models.signals import post_delete
from django.dispatch import receiver
import os

class TemporaryFile(models.Model):
    file = models.FileField(upload_to='temp/')

@receiver(post_delete, sender=TemporaryFile)
def delete_file(sender, instance, **kwargs):
"""حذف الملف عند حذف السجل"""
    if instance.file:
```

```
        if os.path.isfile(instance.file.path):
            os.remove(instance.file.path)
```

3. مهام Cron

bash
```
# crontab -e
# تنظيف الجلسات المنتهية يوميًا الساعة 2 صباحًا
0 2 * * * /path/to/venv/python /path/to/manage.py cleanup_sessions
```

4. مهام Celery الدورية

python
```
# tasks.py
from celery import shared_task
from django.contrib.sessions.models import Session
from django.utils import timezone

@shared_task
def cleanup_expired_sessions():
    expired_sessions = Session.objects.filter(expire_date__lt=timezone.now())
    count = expired_sessions.delete()[0]
    return f'Deleted {count} expired sessions'

# تنظيف كل 24 ساعة
from celery.schedules import crontab

app.conf.beat_schedule = {
    'cleanup-sessions-daily': {
        'task': 'myapp.tasks.cleanup_expired_sessions',
        'schedule': crontab(hour=2, minute=0),
    },
}
```

مقارنة بين الأساليب

حالات الاستخدام	العيوب	المميزات	الأسلوب
تنظيف دوري، مهام صيانة	تتطلب تشغيل يدوي أو cron	سهلة التنفيذ، لا تتتبع تبعيات	**أوامر الإدارة**
تنظيف المرتبط بالسجلات المحذوفة	قد تؤثر على الأداء، صعبة التصحيح	تلقائية، فورية	**الإشارات (Signals)**

حالات الاستخدام	العيوب	المميزات	الأسلوب
المهام الدورية المنتظمة	تعتمد على النظام، لا تتحمل الأخطاء	موثوقة، سهلة الجدولة	**مهامCron**
أنظمة كبيرة، معالجة غير متزامنة	تعقيد الإعداد، تكاليف إضافية	موزعة، تتحمل الأخطاء	**مهامCelery**

أفضل الممارسات

تحسين أداءQuerySet

1. **استخدم** `select_related` **للعلاقات One-to-One و:ForeignKey**

python
```
#جيد
orders = Order.objects.select_related('customer').all()

#سيء
orders = Order.objects.all()  # يؤدي إلى استعلام منفصل لكل طلب
```

2. **استخدم** `prefetch_related` **للعلاقات Many-to-Many و:Reverse ForeignKey**

python
```
#جيد
categories = Category.objects.prefetch_related('products').all()

#سيء
categories = Category.objects.all()  # يؤدي إلى استعلام منفصل لكل فئة
```

3. **استخدم** `only` **و** `defer` **للحد من الحقول المسترجعة**:

python
```
#جيد - جلب الحقول المطلوبة فقط
products = Product.objects.only('name', 'price', 'category')

#سيء - جلب جميع الحقول بما فيها الحقول الكبيرة
products = Product.objects.all()
```

4. **تجنب التقييم المتعدد لنفس:QuerySet**

python
```
#سيء - تقييم مزدوج
products = Product.objects.filter(is_available=True)
if products.exists():  # استعلام أول
    for product in products:  # استعلام ثاني
        print(product.name)

#جيد - تقييم واحد
```

```
products = Product.objects.filter(is_available=True)
products_list = list(products)  # تقييم واحد
if products_list:
    for product in products_list:
        print(product.name)
```

5. **استخدم التجميع في قاعدة البيانات بدلاً من المعالجة في Python:**

python
```
# سيء - المعالجة في Python
total = 0
for product in Product.objects.all():
    total += product.price

# جيد - المعالجة في قاعدة البيانات
from django.db.models import Sum
total = Product.objects.aggregate(Sum('price'))['price__sum']
```

أفضل الممارسات لآلية Clean Up

1. **حدد فترات التنظيف المناسبة**:
   - الجلسات: يوميًا
   - الملفات المؤقتة: كل ساعة
   - سجلات التحقق: أسبوعيًا
2. **استخدم التسجيل المناسب للعمليات**:

python
```
import logging
logger = logging.getLogger(__name__)

def cleanup_task():
    try:
        # عملية التنظيف
        logger.info('Cleanup completed successfully')
    except Exception as e:
        logger.error(f'Cleanup failed: {str(e)}')
```

3. **اختبر عمليات التنظيف في بيئة التطوير أولاً**:
   - تأكد من عمل نسخ احتياطية
   - اختبر على بيانات نموذجية أولاً
4. **راقب أداء عمليات التنظيف**:
   - سجل وقت التنفيذ

- تتبع عدد السجلات المحذوفة
- راجع السجلات بانتظام

سيناريوهات عملية

سيناريو 1: متجر إلكتروني

python

```
#تنظيف عربات التسوق المهجورة
from django.utils import timezone
from datetime import timedelta
from myapp.models import Cart

def cleanup_abandoned_carts():
"""حذف عربات التسوق التي لم يتم تحديثها لأكثر من 30 يومًا"""
    cutoff_date = timezone.now() - timedelta(days=30)
    abandoned_carts = Cart.objects.filter(
        updated_at__lt=cutoff_date,
        is_checked_out=False
    )
    count = abandoned_carts.count()
    abandoned_carts.delete()
    return f'Deleted {count} abandoned carts'
```

سيناريو 2: نظام إدارة المستخدمين

python

```
#تنظيف المستخدمين غير النشطين
from django.contrib.auth.models import User
from django.utils import timezone
from datetime import timedelta

def cleanup_inactive_users():
"""س"""  تعطيل المستخدمين الذين لم يسجلوا دخول لأكثر من سنة
    cutoff_date = timezone.now() - timedelta(days=365)
    inactive_users = User.objects.filter(
        last_login__lt=cutoff_date,
        is_active=True
    )
    count = inactive_users.count()
    inactive_users.update(is_active=False)
    return f'Deactivated {count} inactive users'
```

سيناريو 3: نظام تحميل الملفات

python

```python
# تنظيف الملفات المؤقتة
import os
from django.utils import timezone
from datetime import timedelta
from myapp.models import TemporaryUpload

def cleanup_temp_files():
    """حذف الملفات المؤقتة الأقدم من 24 ساعة"""
    cutoff_date = timezone.now() - timedelta(hours=24)
    old_uploads = TemporaryUpload.objects.filter(created_at__lt=cutoff_date)

    deleted_files = 0
    for upload in old_uploads:
        if os.path.exists(upload.file.path):
            os.remove(upload.file.path)
            deleted_files += 1

    old_uploads.delete()
    return f'Deleted {deleted_files} temporary files'
```

التوصيات

توصيات لاستخدامQuerySet

1. **تفهم التقييم الكسول** :تعلم متى يتم تنفيذ QuerySet لتجنب الاستعلامات غير الضرورية.
2. **استخدم أدوات التصحيح** :استخدم `django-debug-toolbar` لمراقبة استعلامات قاعدة البيانات.
3. **اختبر الأداء** :قم بقياس وقت الاستعلام مع بيانات حقيقية أو مشابهة للبيانات الحقيقية.
4. **استخدم الفهرس المناسب** :أضف indexes للحقول التي تستخدم بشكل متكرر في التصفية والترتيب.

توصيات لآليةClean Up

1. **خطط للتنظيف الدوري** :حدد جدولًا منتظمًا للتنظيف based على احتياجات التطبيق.
2. **استخدم الأسلوب المناسب** :اختر بين الإشارات، أوامر الإدارة، أو المهام الدورية بناءً على متطلباتك.

3. **راجع القوانين واللوائح** :تأكد من أن سياسة الاحتفاظ بالبيانات تتوافق مع القوانين مثل GDPR.
4. **أنشئ نسخ احتياطية** :دائماً احتفظ بنسخ احتياطية قبل عمليات التنظيف الكبيرة.
5. **وثق العمليات** :سجل جميع عمليات التنظيف وأي أخطاء تحدث لسهولة المراجعة والتصحيح.

أدوات مساعدة مقترحة

1. **django-extensions**: يوفر أوامر إدارة إضافية مثل `shell_plus` و`runserver_plus`.
2. **django-debug-toolbar**: أداة ضرورية لتصحيح وتحسين استعلامات قاعدة البيانات.
3. **django-cleanup**: حزمة تلقائية لحذف الملفات عند حذف النماذج.
4. **celery**: للمهام الدورية والغير متزامنة في التطبيقات الكبيرة.

خاتمة :يعد فهم آلية عمل QuerySet وآلية Clean Up في Django أساسياً لبناء تطبيقات ويب فعالة وقابلة للصيانة. من خلال تطبيق أفضل الممارسات المذكورة في هذا البحث، يمكن للمطورين تحسين أداء تطبيقاتهم، والحفاظ على نظافة البيانات، وضمان تجربة مستخدم ممتازة.