Django Model Nesneleri Üzerinde CRUD

Gerekli tahmini süre: 30 dakika

Bu laboratuvar çalışmasında, ilişkili Django Modelleri oluşturma, nesneleri veritabanlarına kaydetme, sorgulama, güncelleme ve silme pratiği yapacaksınız.

Öğrenme Hedefleri

- Bir-Bir, Bir-Çok ve Çok-Çok ilişkileri ile Django Modelleri oluşturun
- Filtreler ile model nesnelerini sorgulayın
- Nesneleri silin ve güncelleyin

Bulut IDE'de Dosyalarla Çalışma

Eğer Bulut IDE'ye yeniyseniz, bu bölüm projenizin bir parçası olan dosyaları nasıl oluşturup düzenleyeceğinizi gösterecektir.

Bulut IDE içindeki dosyalarınızı ve dizinlerinizi görüntülemek için bu dosya simgesine tıklayın.

`Yeni`ye tıklayın, ardından `Yeni Terminal` seçeneğine tıklayın.

Bu, komutlarınızı çalıştırabileceğiniz yeni bir terminal açacaktır.

Laboratuvarda Kapsanan Kavramlar

1. `CRUD operations`: Oluşturma, Okuma, Güncelleme ve Silme işlemleri
2. `QuerySet`: Bir veritabanındaki kayıtların bir koleksiyonunu temsil eder ve Django Model API'sini kullanarak nesneleri okumak için gereklidir.
3. `all()` method: Veritabanındaki tüm nesnelerin bir QuerySet'ini döndürür.
4. `filter()` method: Sadece arama terimiyle eşleşen satırları döndürür. **büyüktür**, **küçüktür**, **içerir** veya **nulldır** gibi arama parametreleri olabilir.

Theia'da PostgreSQL Başlatma

- PostgreSQL'ü, sol menü çubuğundaki SkillsNetwork simgesini bulup DATABASES menü öğesinden PostgreSQL'i seçerek UI'dan başlatın:
- PostgreSQL başlatıldıktan sonra, sunucu bağlantı bilgilerini UI'dan kontrol edebilirsiniz. Django uygulamasının bu veritabanına bağlanması için kullanılacak `username`, `password` ve `host` gibi bağlantı bilgilerini markdown formatında yazın.
- Ayrıca bir `pgAdmin` örneğinin kurulu ve başlatılmış olduğunu göreceksiniz.

Bağımsız Django ORM proje şablonunu içe aktarma

Laboratuvara başlamadan önce, mevcut Theia dizininizin `/home/project` olduğundan emin olun.

Öncelikle, bir sanal ortam oluşturmalı ve Django ile psycopg ile ilgili paketleri kurmalıyız.

- Terminal açık değilse, `Terminal > Yeni Terminal` seçeneğine gidin ve şunu çalıştırın:

```
pip install --upgrade distro-info
pip3 install --upgrade pip==23.2.1
pip install virtualenv
virtualenv djangoenv
source djangoenv/bin/activate
pip install Django==4.2 psycopg2-binary
```

- Bu laboratuvar için bir kod şablonu indirmek üzere aşağıdaki komut satırlarını çalıştırın.

```
wget "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-CD0251EN-SkillsNetwork/labs/m3_django_orm/lab2_template.zip"
unzip lab2_template.zip
rm lab2_template.zip
```

Projeniz aşağıdaki gibi görünmelidir:

- `settings.py` dosyasını açın ve `DATABASES` bölümünü bulun. `PASSWORD` değerini oluşturulan `PostgreSQL` parolası ile değiştirin.

Artık `settings.py` dosyanız aşağıdaki gibi görünmelidir:

Çevrimiçi Kurs Uygulaması İçin Modeller Oluşturma

Sonra, çevrimiçi bir kurs uygulaması için modeller oluşturmaya başlayabilirsiniz.

- `crud/models.py` dosyasını açın ve ilk `User` modelini ekleyin.

```
# User model
class User(models.Model):
    first_name = models.CharField(null=False, max_length=30, default='john')
    last_name = models.CharField(null=False, max_length=30, default='doe')
    dob = models.DateField(null=True)

    # Create a toString method for object string representation
    def __str__(self):
        return self.first_name + " " + self.last_name
```

The `User` modeli, `first_name`, `last_name` gibi `CharField` türünde ve `dob` olarak `DateField` türünde bir kullanıcının ortak bilgilerini içerir.

Buna ek olarak, bir kullanıcı nesnesinin string temsilini oluşturmak için `__str__(self):` metodunu geçersiz kılıyoruz. Bu, bir kullanıcı nesnesini yazdırmak istediğinizde kullanışlıdır.

Ayrıca, `User` modeline `email` veya `location` gibi istediğiniz kadar ilkel alan eklemekten çekinmeyin. Model alan tanımları hakkında daha fazla bilgi bulabilirsiniz: Django Model Fields

Sonraki adımda, `User` modelinden miras alınan bir `Instructor` modeli ekleyelim ve bunu `One-To-One` ilişkisi olarak tanımlayalım. `Instructor`, `full_time` ve `total_learners` gibi daha fazla eğitmen spesifik alan ekleyen bir `User` uzantısıdır.

- `models.py` dosyasına bir `Instructor` modeli ekleyin.

```
# Instructor model
class Instructor(User):
    full_time = models.BooleanField(default=True)
    total_learners = models.IntegerField()

    # Create a toString method for object string representation
    def __str__(self):
        return "First name: " + self.first_name + ", " + \
               "Last name: " + self.last_name + ", " + \
               "Is full time: " + str(self.full_time) + ", " + \
               "Total Learners: " + str(self.total_learners)
```

O halde, `instructor` modeli ile `Many-To-Many` ilişkisi olan bir `Course` modeli oluşturalım, bu ilişki `instructors` referans alanı ile tanımlanır.

- `models.py` dosyasına bir `Course` modeli ekleyin.

```
# Course model
class Course(models.Model):
    name = models.CharField(null=False, max_length=100, default='online course')
    description = models.CharField(max_length=500)
    # Many-To-Many relationship with Instructor
    instructors = models.ManyToManyField(Instructor)

    # Create a toString method for object string representation
    def __str__(self):
        return "Name: " + self.name + "," + \
            "Description: " + self.description
```

Burada `Course` ve `Instructor` arasında bir Many-To-Many ilişkisi ekledik ve `instructors` adında bir `ManyToManyField` alanı oluşturduk.

Bir kurs genellikle birkaç dersten oluşur ve bu nedenle `Lesson` modeline karşı `One-To-Many` ilişkisi vardır; yani, her kursun sıfır veya birçok dersi olabilir, ancak her ders yalnızca bir kursa aittir.

- `models.py` dosyasına bir `Lesson` modeli ekleyin.

```
# Lesson
class Lesson(models.Model):
    title = models.CharField(max_length=200, default="title")
    course = models.ForeignKey(Course, null=True, on_delete=models.CASCADE)
    content = models.TextField()
```

Kodlama Pratiği: Bir Öğrenci Modeli Ekle

Aşağıdaki kod parçacığını, `User`'dan miras alan ve bazı öğrenci ile ilgili alanlara sahip bir `Learner` modeli eklemek için tamamlayın:

1. Eksik kısımlar için `<HINT>` ile birlikte yorumlara bakabilirsiniz.
2. `Course` modelinden önce `models.py` dosyasında `Learner` modelini tanımlamanız gerekiyor, böylece `Course` modeli `Learner`'ın varlığını bilebilir.
3. Değişikliklerin etkili olması için güncellenmiş dosyaları kaydetmeyi unutmayın.

```
# Learner model
class Learner(##<HINT> add a user parent model):
    STUDENT = 'student'
    DEVELOPER = 'developer'
    DATA_SCIENTIST = 'data_scientist'
    DATABASE_ADMIN = 'dba'
    OCCUPATION_CHOICES = [
        (STUDENT, 'Student'),
        (DEVELOPER, 'Developer'),
        (DATA_SCIENTIST, 'Data Scientist'),
        (DATABASE_ADMIN, 'Database Admin')
    ]
    occupation = models.CharField(
        null=False,
        max_length=20,
        choices=OCCUPATION_CHOICES,
        default=STUDENT
    )
    social_link = models.URLField(max_length=200)

    ##<HINT> Create a __str__ method returning a string presentation
    def __str__(self):
        ...
```

▼ Çözümü görmek için buraya tıklayın

```
# Learner model
class Learner(User):
    STUDENT = 'student'
    DEVELOPER = 'developer'
    DATA_SCIENTIST = 'data_scientist'
    DATABASE_ADMIN = 'dba'
    OCCUPATION_CHOICES = [
        (STUDENT, 'Student'),
        (DEVELOPER, 'Developer'),
        (DATA_SCIENTIST, 'Data Scientist'),
        (DATABASE_ADMIN, 'Database Admin')
    ]
    # Occupation Char field with defined enumeration choices
    occupation = models.CharField(
        null=False,
        max_length=20,
        choices=OCCUPATION_CHOICES,
        default=STUDENT
```

```
    )
    # Social link URL field
    social_link = models.URLField(max_length=200)

    # Create a toString method for object string representation
    def __str__(self):
        return "First name: " + self.first_name + ", " + \
               "Last name: " + self.last_name + ", " \
               "Date of Birth: " + str(self.dob) + ", " + \
               "Occupation: " + self.occupation + ", " + \
               "Social Link: " + self.social_link
```

`occupation` alanı için, mesleklerin değerlerini sınırlamak amacıyla bir sıralı seçim ekledik.

::page{title="Kodlama Pratiği: Bir Kayıt Modeli Ekle"}

Aşağıdaki `Enrollment` sınıfını ekleyin.
`Course` modelini, `Enrollment` sınıfı aracılığıyla `Learner` modeli ile `Çoktan-Çoka` bir ilişki ekleyecek şekilde güncelleyin.

```
# Enrollment model as a lookup table with additional enrollment info
class Enrollment(models.Model):
    AUDIT = 'audit'
    HONOR = 'honor'
    COURSE_MODES = [
        (AUDIT, 'Audit'),
        (HONOR, 'Honor'),
    ]
    # Add a learner foreign key
    learner = models.ForeignKey(Learner, on_delete=models.CASCADE)
    # Add a course foreign key
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    # Enrollment date
    date_enrolled = models.DateField(default=now)
    # Enrollment mode
    mode = models.CharField(max_length=5, choices=COURSE_MODES, default=AUDIT)
```

▶ Çözümü görmek için buraya tıklayın

::page{title="Modelleri Taşı"}

Artık şunları tanımladınız:

1. Bir-Bir ilişkisi ile `User` ve `Instructor` modelleri
2. Bir-Çok ilişkisi ile `Course` ve `Lessons` modelleri
3. Çok-Çok ilişkisi ile `Course` ve `Instructor` modelleri.

`crud` uygulaması için bu tabloları `PostgreSQL` veritabanımızda oluşturmak üzere göçleri çalıştıralım.

- Mevcut çalışma dizininiz `/home/project/lab2_template` değilse, proje klasörüne `cd` komutunu kullanın.

  ```
  cd lab2_template
  ```

- Ardından `crud` uygulaması için göç betikleri oluşturun.

  ```
  python3 manage.py makemigrations crud
  ```

ve Django'nun aşağıdaki tabloları oluşturmak üzere olduğunu görmelisiniz.

```
Migrations for 'crud':
  crud/migrations/0001_initial.py
    - Create model Course
    - Create model User
    - Create model Instructor
    - Create model Learner
    - Create model Lesson
    - Create model Enrollment
    - Add field instructors to course
    - Add field learners to course
```

- Göçleri çalıştır.

```
python3 manage.py migrate
```

ve `crud.0001_initial` adlı göç scriptinin çalıştırıldığını görmelisiniz.

```
Operations to perform:
  Apply all migrations: crud
Running migrations:
  Applying crud.0001_initial... OK
```

::page{title="Nesne Oluşturma ve Silme"}

Bu noktada, bu laboratuvar için tüm modelleri tanımladınız.

Bu modeller üzerinde bazı oluşturma ve silme işlemleri gerçekleştirmeyi deneyelim.

- `write.py` dosyasını açın ve bazı eğitmen nesnelerini kaydetmek için bir `write_instructors()` yöntemi ekleyin.

```
def write_instructors():
    # Add instructors
    # Create a user
    user_john = User(first_name='John', last_name='Doe', dob=date(1962, 7, 16))
    user_john.save()
    instructor_john = Instructor(full_time=True, total_learners=30050)
    # Update the user reference of instructor_john to be user_john
    instructor_john.user = user_john
    instructor_john.save()

    instructor_yan = Instructor(first_name='Yan', last_name='Luo', dob=date(1962, 7, 16), full_time=True, total_learners=30050)
    instructor_yan.save()
    instructor_joy = Instructor(first_name='Joy', last_name='Li', dob=date(1992, 1, 2), full_time=False, total_learners=10040)
    instructor_joy.save()
    instructor_peter = Instructor(first_name='Peter', last_name='Chen', dob=date(1982, 5, 2), full_time=True, total_learners=2002)
    instructor_peter.save()
    print("Instructor objects all saved... ")
```

`instructor_john` için önce onun üst sınıf modeli `user`'ı oluşturuyoruz ve `instructor_john.user`'ı `user_john` olarak güncelliyoruz.

Diğer eğitmenler için Django, `first_name`, `last_name`, `dob` değerlerini otomatik olarak üst kullanıcı nesnelerine atayacaktır.

- Bazı kurs nesneleri eklemek için `write_courses()` yöntemini ekleyin.

```
def write_courses():
    # Add Courses
```

```
    course_cloud_app = Course(name="Cloud Application Development with Database",
                              description="Develop and deploy application on cloud")
    course_cloud_app.save()
    course_python = Course(name="Introduction to Python",
                           description="Learn core concepts of Python and obtain hands-on "
                                       "experience via a capstone project")
    course_python.save()
    print("Course objects all saved... ")
```

- Bazı dersler eklemek için bir `write_lessons()` yöntemi ekleyin

```
def write_lessons():
    # Add lessons
    lession1 = Lesson(title='Lesson 1', content="Object-relational mapping project")
    lession1.save()
    lession2 = Lesson(title='Lesson 2', content="Django full stack project")
    lession2.save()
    print("Lesson objects all saved... ")
```

- Veritabanı tablolarınızı pratik bir şekilde temizlemek için aşağıdaki kod parçasına benzer bir `clean_data()` metodu ekleyebilirsiniz. Bu, önce tüm nesneleri almak için model yöneticisi `objects` kullanır ve ardından bunları veritabanından siler.

```
def clean_data():
    # Delete all data to start from fresh
    Enrollment.objects.all().delete()
    User.objects.all().delete()
    Learner.objects.all().delete()
    Instructor.objects.all().delete()
    Course.objects.all().delete()
    Lesson.objects.all().delete()
```

Sonra, nesneleri gerçekten kaydetmek için bu doldurma yöntemlerini çağıralım

- Aşağıdaki yöntem çağrılarını `write.py` dosyasına ekleyin

```
# Clean any existing data first
clean_data()
write_courses()
write_instructors()
write_lessons()
```

- Sonunda, terminalde `write.py` dosyasını çalıştırın.

```
python3 write.py
```

Terminalde kaydedilen tüm mesajların nesnelerini görmelisiniz; bu, kaydetme işlemlerinin başarıyla gerçekleştirildiğini gösterir.

```
    Course objects all saved...
```

```
Instructor objects all saved...
Lesson objects all saved...
```

Sonraki adımda, kaydedilmiş nesneleri sorgulamayı deneyelim.

::page{title="Kodlama Pratiği: Daha Fazla Öğrenici Nesnesi Oluşturma ve Kaydetme"}

Aşağıdaki kod parçasını `write_learners()` yöntemi ile tamamlayarak daha fazla öğrenici nesnesini veritabanına kaydedin:

1. Eksik kısımlar için `<HINT>` ile belirtilen yorumlara bakabilirsiniz.

```
def write_learners():
    # Add Learners
    learner_james = Learner(first_name='James', last_name='Smith', dob=date(1982, 7, 16),
                            occupation='data_scientist',
                            social_link='https://www.linkedin.com/james/')
    learner_james.save()
    learner_mary = Learner(first_name='Mary', last_name='Smith', dob=date(1991, 6, 12), occupation='dba',
                           social_link='https://www.facebook.com/mary/')
    learner_mary.save()
    learner_robert = Learner(first_name='Robert', last_name='Lee', dob=date(1999, 1, 2), occupation='student',
                             social_link='https://www.facebook.com/robert/')
    learner_robert.save()
    learner_david = Learner(first_name='David', last_name='Smith', dob=date(1983, 7, 16),
                            occupation='developer',
                            social_link='https://www.linkedin.com/david/')
    learner_david.save()
    learner_john = Learner(first_name='John', last_name='Smith', dob=date(1986, 3, 16),
                           occupation='developer',
                           social_link='https://www.linkedin.com/john/')
    learner_john.save()
    print("Learner objects all saved... ")
    #<HINT> Add more learners objects#
    #...
```

2. Add the `write_learners()` method call to `write.py`

```
# Önce mevcut verileri temizle
clean_data()
write_courses()
write_instructors()
write_lessons()
write_learners()
```

3. Run the `write.py` file in terminal again

```
python3 write.py
```

::page{title="Query objects"}

We first read all `courses`.

- Open `read_courses.py` and add the following code snippet:

```
#Tüm kursları bul
courses = Course.objects.all()
print(courses)
```

In the above code snippet, we call the model managers `objects` to return us all the courses.

- Let's run the Python script file to test the result

```
`python3 read_courses.py`
```

You should see a `QuerySet` object containing the two courses we created in the previous step.

```
<QuerySet [
<Course: Name: Bulut Uygulama Geliştirme ile Veritabanı,Description: Bulutta uygulama geliştirin ve dağıtın>,
<Course: Name: Python'a Giriş,Description: Python'un temel kavramlarını öğrenin ve bir capstone projesi aracılığıyla uygulamalı deneyim kazanın>
]>
```

- Next, let's query instructors with filters to select subsets of instructors meeting following criterions:

1. Find a single instructor with first name `Yan`
2. Try to find a non-existing instructor with first name `Andy`
3. Find all part time instructors
4. Find all full time instructors with First Name starts with `Y` and learners count greater than 30000
5. Find all full time instructors with First Name starts with `Y` and learners count greater than 30000 using multiple parameters

- Open `read_instructor.py`, and add the following code snippets to perform queries:

```
```python
instructor_yan = Instructor.objects.get(first_name="Yan")
print("1. `Yan` adında bir eğitmeni bul")
print(instructor_yan)
print("\n")
# `Andy` adında bir eğitmen bulunmadığını unutmayın
# Bu nedenle yönetici bir istisna fırlatacaktır
try:
    instructor_andy = Instructor.objects.get(first_name="Andy")
except Instructor.DoesNotExist:
    print("2. `Andy` adında mevcut olmayan bir eğitmeni bulmaya çalış")
    print("Eğitmen Andy mevcut değil")
print("\n")
part_time_instructors = Instructor.objects.filter(full_time=False)
print("3. Tüm yarı zamanlı eğitmenleri bul: ")
print(part_time_instructors)
print("\n")
full_time_instructors = Instructor.objects.exclude(full_time=False).filter(total_learners__gt=30000).\
        filter(first_name__startswith='Y')
print("4. `Y` ile başlayan ve öğrenci sayısı 30000'den fazla olan tüm tam zamanlı eğitmenleri bul")
print(full_time_instructors)
print("\n")
full_time_instructors = Instructor.objects.filter(full_time=True, total_learners__gt=30000,
                                                  first_name__startswith='Y')
print("5. `Y` ile başlayan ve öğrenci sayısı 30000'den fazla olan tüm tam zamanlı eğitmenleri bul")
print(full_time_instructors)
```

```
*<b> Review the above code examples to understand how each filter and parameters were made. </b>*
- Run `read_instructors.py` in the terminal
```
```

python3 read_instructors.py

```
Query results:
```

1. `Yan` adında bir eğitmen bulun
   Ad: Yan, Soyad: Luo, Tam zamanlı mı: Evet, Toplam Öğrenci: 30050
2. `Andy` adında var olmayan bir eğitmen bulmaya çalış
   Eğitmen Andy mevcut değil
3. Tüm yarı zamanlı eğitmenleri bulun:
   <QuerySet [<Instructor: Ad: Joy, Soyad: Li, Tam zamanlı mı: Hayır, Toplam Öğrenci: 10040>]>
4. `Y` harfi ile başlayan ve öğrenci sayısı 30000'den fazla olan tüm tam zamanlı eğitmenleri bulun
   <QuerySet [<Instructor: Ad: Yan, Soyad: Luo, Tam zamanlı mı: Evet, Toplam Öğrenci: 30050>]>
5. `Y` harfi ile başlayan ve öğrenci sayısı 30000'den fazla olan tüm tam zamanlı eğitmenleri bulun
   <QuerySet [<Instructor: Ad: Yan, Soyad: Luo, Tam zamanlı mı: Evet, Toplam Öğrenci: 30050]>

````
::page{title="Coding practice: Query Learners with Filters"}
Open `read_learners.py`, complete and append the code snippet to query subset learners based on the following criterions:
  1. Find learners with last name `Smith`
  2. Find two youngest learners (ordered by `dob`)

```python
 # "Smith" soyadına sahip öğrencileri bul
learners_smith = Learner.objects.filter(#<HINT> soyad kontrolü ekle)
print("1. `Smith` soyadına sahip öğrencileri bul")
print(learners_smith)
print("\n")
# Doğum tarihine göre azalan sırada sırala ve ilk iki nesneyi seç
learners = Learner.objects.order_by(#<HINT> doğum tarihini - ile azalan olarak ekle )[#<HINT> indeks 0:2 ekle]
print("2. En genç iki öğrenciyi bul")
print(learners)
````

▼ Click here to see solution

```
 # Soyadı "Smith" olan öğrencileri bul
learners_smith = Learner.objects.filter(last_name="Smith")
print("1. Soyadı `Smith` olan öğrencileri bul:")
print(learners_smith)
print("\n")
# Doğum tarihine göre azalan sıralama yap ve ilk iki nesneyi seç
learners = Learner.objects.order_by('-dob')[0:2]
print("2. En genç iki öğrenciyi bul:")
print(learners)
```

- Run the `read_learners.py` file:

  ```
  `python3 read_learners.py`
  ```
  ```

Query results:

```

1. Soyadı `Smith` olan öğrenicileri bul
   <QuerySet [<Learner: Ad: James, Soyad: Smith, Doğum Tarihi: 1982-07-16, Meslek: veri_bilimci, Sosyal Bağlantı: https://www.linkedin.com/james/>,
   <Learner: Ad: Mary, Soyad: Smith, Doğum Tarihi: 1991-06-12, Meslek: dba, Sosyal Bağlantı: https://www.facebook.com/mary/>,
   <Learner: Ad: David, Soyad: Smith, Doğum Tarihi: 1983-07-16, Meslek: geliştirici, Sosyal Bağlantı: https://www.linkedin.com/david/>,
   <Learner: Ad: John, Soyad: Smith, Doğum Tarihi: 1986-03-16, Meslek: geliştirici, Sosyal Bağlantı: https://www.linkedin.com/john/>]>

2. En genç iki öğreniciyi bul
   <QuerySet [<Learner: Ad: Robert, Soyad: Lee, Doğum Tarihi: 1999-01-02, Meslek: öğrenci, Sosyal Bağlantı: https://www.facebook.com/robert/>,
   <Learner: Ad: Mary, Soyad: Smith, Doğum Tarihi: 1991-06-12, Meslek: dba, Sosyal Bağlantı: https://www.facebook.com/mary/>]>
```

::page{title="Özet"}

Bu laboratuvar çalışmasında, bağımsız bir Django ORM proje şablonunu içe aktardınız. Şablona dayanarak, ilişkili Django Modelleri oluşturmayı, bu Django Model nesnelerini veritabanlarına kaydetmeyi, güncelleme ve silme işlemleri yapmayı ve bunları filtrelerle sorgulamayı öğrendiniz ve pratik yaptınız.

Sonraki adımda, ilişkili nesnelere nasıl erişeceğinizi öğreneceksiniz.

Yazar(lar)

Yan Luo

©IBM Corporation. Tüm hakları saklıdır.