# **Lab Assignment: Step by Step Guide**

# Practice Problems for Object-Relational Mapping

# Learning Objectives

1. Create Django Models with One-To-One, One-To-Many, and Many-To-Many relationships
2. Create and delete objects
3. Query model objects with filters

# Create Models for below Online Course App

Consider the below high-level model of online course app. Create relevant classes and perform CRUD operation using Django ORM APIs

Diagram

Description automatically generated

Many-To-One

# Create Django Project

Go to VS Code termina and start a new project under your Django Project

*django-admin startproject OnlineCourse*

change directory to OnlineCourse and create new application for this project

***python manage.py startapp CRUD***

Text

Description automatically generated

***Register CRUD app in installed applications of OnlineCourse ‘setting.py’ file***

Text

Description automatically generated

Create Models

Next, you can start to create models for an online course app.

Open *crud/models.py* and append first User model

**# User model**

# 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 model contains common information about a user such as first\_name, last\_name as CharField and dob as DateField.

In addition to that, we override the \_\_str\_\_(self): method to create a string representation of a user object. This is convenient if you want to print a user object.

*Also feel free to add as many primitive fields as you like to the User model such as email or location. You could find more details about model field definitions here*[Django Model Fields](https://docs.djangoproject.com/en/3.1/ref/models/fields/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMCD0251ENSkillsNetwork21879158-2021-01-01)*:*

[Model field reference | Django documentation | Django (djangoproject.com)](https://docs.djangoproject.com/en/3.1/ref/models/fields/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMCD0251ENSkillsNetwork21879158-2021-01-01)

Next, let's add an Instructor model inherited from User model and make it as One-To-One relationship. Instructor is an extension of User which adds some more instructor specific fields such as full\_time and total\_learners.

Append an Instructor model to models.py

# 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)

Then, let's create a Course model which has a Many-To-Many relationship to Instructor model, defined by the reference field instructors.

Append a Course model to models.py

# 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

Here we added a Many-To-Many relationship between **Course** and **Instructor** by creating a ManyToManyField called instructors

A course normally contains several lessons thus has a One-To-Many relationship to a Lesson model, i.e., each course can have zero or many lessons but each lesson only belongs to one course.

Append a Lesson model to models.py

# 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()

Coding Practice: Add a Learner Model

Complete the following code snippet to add a Learner model inherited from User with some learner related fields:

You need to define Learner model before Course model in models.py so that the Course model knows the existence of Learner

Remember to save the updated files to make the changes effective.

# 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):

        ...

Note for the occupation field, we added an enumerate choices for limiting the values of occupations

# Solution:

Solution should go here

Coding Practice: Add an Enrollment Model

Append the following Enrollment class and update Course model to add a Many-To-Many relationship with Learner model via the Enrollment class

# 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)

# Solution:

Update Course model by adding learners reference field similar to instructors reference field

# Update 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 Instructors

    instructors = models.ManyToManyField(Instructor)

# Complete below code for Many-To-Many relationship with Learner via Enrollment relationship, similar to instructors above

#Hint: learner should be referenced to enrolment through look up object via keyword ‘through’ e.g. through=’enrollment’. Complete your code for learner below:

    learners =

    def \_\_str\_\_(self):

        return "Name: " + self.name + "," + \

                 "Description: " + self.description

To summarize the relationships of these models:

* Learner and Instructor models are inherited from User model with One-To-One relationship
* Lesson has One-To-Many relationship with Course
* Instructor has Many-To-Many with Course
* Learner has Many-To-Many with Course model via Enrollment

# Migrate Models

Now, you have defined User and Instructor models with One-To-One relationship,

Course and Lessons models with One-To-Many relationship, and

Course and Instructor with Many-To-Many relationship.

Let's run migrations for the CRUD app to create those tables in our SQLite database.

If your current working directory is not set to current project (in this case OnlineCourse) change directory to current project folder:

* Use following command to see what is ready to migrate:

*Python manage.py showmigrations*

PS C:\DjangoProjects\OnlineCourse> python *manage.py showmigrations*

Text

Description automatically generated

* Then generate migration scripts for app CRUD

*python manage.py makemigrations*

Text

Description automatically generated

You should see migration script crud.0001\_initial under migration folder.

Text

Description automatically generated

*To show SQL statement for the model use run following command*

***python manage.py sqlmigrate <application name, migration file name>***

***python manage.py sqlmigrate CRUD 0001***

Text

Description automatically generated

* Finally, run the migration

*python manage.py migrate*

Text

Description automatically generated

*Created table now can be viewed in SQLite database:*

*Go to open database and navigate to db.sqlite3 folder from the OnlineCourse folder*

*Note the table name and schema*

Table

Description automatically generated with low confidence

Note we have created 6 classes but sqlite database shows 7 table. This is because many-to-many relationships are mapped into new tables with key from the base table.

In this case we have many-to-many relationship with course and instructor, that is why an extra table <CRUD\_course\_instructor> is created.

# Create and Delete Objects

At this point, you have defined all models for this lab, let's try to perform some create and delete operations on those models.

Create a new file ‘write.py’ and use below code to connect with your model

# Django specific settings

import inspect

import os

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'OnlineCourse.settings')

from django.db import connection

# Ensure settings are read

from django.core.wsgi import get\_wsgi\_application

application = get\_wsgi\_application()

from CRUD.models import \*

from datetime import date

and write\_instructors() method to save some instructors objects.

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... ")

For the instructor\_john, we first create his parent class model user and update instructor\_john.user to be user\_john.

For other instructors, Django will automatically assign values of first\_name, last\_name, dob to their parent user objects.

* Append a write\_courses() method to add some course objects.

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("Learners objects saved... ")

* Append a write\_courses() method to add some course objects.

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... ")

* Append a write\_lessons() method to add some lessons

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... ")

* Append a write\_course\_enrollment\_relationships():method to add students to courses

def write\_course\_enrollment\_relationships():

    # Get related courses

    course\_cloud\_app = Course.objects.get(name\_\_contains='Cloud')

    course\_python = Course.objects.get(name\_\_contains='Python')

    # Get related learners

    learner\_james = Learner.objects.get(first\_name='James')

    learner\_mary = Learner.objects.get(first\_name='Mary')

    learner\_david = Learner.objects.get(first\_name='David')

    learner\_john = Learner.objects.get(first\_name='John')

    learner\_robert = Learner.objects.get(first\_name='Robert')

    # Add enrollments

    james\_cloud = Enrollment.objects.create(learner=learner\_james, date\_enrolled=date(2020, 8, 1),

                                            course=course\_cloud\_app, mode='audit')

    james\_cloud.save()

    mary\_cloud = Enrollment.objects.create(learner=learner\_mary, date\_enrolled=date(2020, 8, 2),

                                         course=course\_cloud\_app, mode='honor')

    mary\_cloud.save()

    david\_cloud = Enrollment.objects.create(learner=learner\_david, date\_enrolled=date(2020, 8, 5),

                                            course=course\_cloud\_app, mode='honor')

    david\_cloud.save()

    david\_cloud = Enrollment.objects.create(learner=learner\_john, date\_enrolled=date(2020, 8, 5),

                                           course=course\_cloud\_app, mode='audit')

    david\_cloud.save()

    robert\_python = Enrollment.objects.create(learner=learner\_robert, date\_enrolled=date(2020, 9, 2),

                                              course=course\_python, mode='honor')

    robert\_python.save()

    print("Course-learner relationships saved... ")

# Coding Practice to add instructors for courses

* Append a write\_course\_instructor\_relationships() method to add courses for instructors

def write\_course\_instructor\_relationships():

    # Get related instructors

    # Get related courses

    # Add instructors to courses

    print("Course-instructor relationships saved... ")

* To conveniently clean up your database tables, you can add a clean\_data() method like the following code snippet.
* It uses the model manager objects to get all objects first and then delete them from database

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()

Next, let's call those populating methods to actually save the objects

* Append the following methods call to write.py

# Clean any existing data first

clean\_data()

write\_courses()

write\_instructors()

write\_lessons()

write\_lessons()

write\_course\_enrollment\_relationships()

write\_course\_instructor\_relationships()

Run the **write.py** in terminal

python write.py

You should see objects and relationships saved messages in terminal.

Course objects all saved...

Instructor objects all saved...

Learners objects saved...

Lesson objects all saved...

Course-instructor relationships saved...

Course-learner relationships saved...

Verify records in SQLite, you will see data is populated in the tables

Graphical user interface, text, application, email

Description automatically generated

# Query objects

We first read all courses.

* create read\_courses.py and add the following code snippet:

# Django specific settings

import inspect

import os

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'OnlineCourse.settings')

from django.db import connection

# Ensure settings are read

from django.core.wsgi import get\_wsgi\_application

application = get\_wsgi\_application()

from CRUD.models import \*

from datetime import date

#Find all courses

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 results

Python read\_courses.py

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

<QuerySet [

<Course: Name: Cloud Application Development with Database,Description: Develop **and** deploy application on cloud>,

<Course: Name: Introduction to Python,Description: Learn core concepts **of** Python **and** obtain hands-on experience via a capstone project>

]>

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

Create read\_instructor.py, and add the following code snippets to perform queries

# Django specific settings

import inspect

import os

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'OnlineCourse.settings')

from django.db import connection

# Ensure settings are read

from django.core.wsgi import get\_wsgi\_application

application = get\_wsgi\_application()

from CRUD.models import \*

from datetime import date

instructor\_yan = Instructor.objects.get(first\_name="Yan")

print("1. Find a single instructor with first name `Yan`")

print(instructor\_yan)

print("\n")

# Note that there is no instructor with first name `Andy`

# So the manager will throw an exception

try:

    instructor\_andy = Instructor.objects.get(first\_name="Andy")

except Instructor.DoesNotExist:

    print("2. Try to find a non-existing instructor with first name `Andy`")

    print("Instructor Andy doesn't exist")

print("\n")

part\_time\_instructors = Instructor.objects.filter(full\_time=False)

print("3. Find all part time instructors: ")

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. Find all full time instructors with First Name starts with `Y` and learners count greater than 30000")

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. Find all full time instructors with First Name starts with `Y` and learners count greater than 30000")

print(full\_time\_instructors)

Review the above code examples to understand how each filter and parameters were made.

* Run read\_instructors.py in the terminal

**Query Results: you should see the following output**

1. Find a single instructor with first name `Yan`

First name: Yan, Last name: Luo, Is full time: True, Total Learners: 30050

2. Try to find a non-existing instructor with first name `Andy`

Instructor Andy doesn't exist

3. Find all part time instructors:

<QuerySet [<Instructor: First name: Joy, Last name: Li, Is full time: False, Total Learners: 10040>]>

4. Find all full time instructors with First Name starts with `Y` and learners count greater than 30000

<QuerySet [<Instructor: First name: Yan, Last name: Luo, Is full time: True, Total Learners: 30050>]>

5. Find all full time instructors with First Name starts with `Y` and learners count greater than 30000

<QuerySet [<Instructor: First name: Yan, Last name: Luo, Is full time: True, Total Learners: 30050>]>

# Coding practice: Query Learners with Filters

Create read\_learners.py, complete and append the code snippet to query subset learners based on the following criterions:

Find learners with last name Smith

Find two youngest learners (ordered by dob)

# Django specific settings

import inspect

import os

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'OnlineCourse.settings')

from django.db import connection

# Ensure settings are read

from django.core.wsgi import get\_wsgi\_application

application = get\_wsgi\_application()

from CRUD.models import \*

from datetime import date

 # Find students with last name "Smith"

learners\_smith = Learner.objects.filter(#<HINT> add last\_name check)

print("1. Find learners with last name `Smith`")

print(learners\_smith)

print("\n")

# Order by dob descending, and select the first two objects

learners = Learner.objects.order\_by(#<HINT> add dob with - as descending )[#<HINT> add index 0:2]

print("2. Find top two youngest learners")

print(learners)

Run the read\_learners.py file in the terminal

You should see following output in the terminal after completing the code

1. Find learners with last name `Smith`:

<QuerySet [

<Learner: First name: James, Last name: Smith, Date of Birth: 1982-07-16, Occupation: data\_scientist, Social Link: https://www.linkedin.com/james/>,

<Learner: First name: Mary, Last name: Smith, Date of Birth: 1991-06-12, Occupation: dba, Social Link: <https://www.facebook.com/mary/>>,

<Learner: First name: David, Last name: Smith, Date of Birth: 1983-07-16, Occupation: developer, Social Link: https://www.linkedin.com/david/>,

<Learner: First name: John, Last name: Smith, Date of Birth: 1986-03-16, Occupation: developer, Social Link: <https://www.linkedin.com/john/>>]>

2. Find top two youngest learners:

<QuerySet [

<Learner: First name: Robert, Last name: Lee, Date of Birth: 1999-01-02, Occupation: student, Social Link: <https://www.facebook.com/robert/>>,

<Learner: First name: Mary, Last name: Smith, Date of Birth: 1991-06-12, Occupation: dba, Social Link: https://www.facebook.com/mary/>]

# Making querying span relationships

In this section, you will focus on querying objects spanning relationships. For example, get all instructors for a course or get learners enrolled in a particular course.

* Let's start with querying objects across relationships for following scenarios:

1. Get courses taught by Instructor Yan, via both forward (explicit) and backward (implicit) access
2. Get the instructors of Cloud app dev course
3. Check the occupations of the courses taught by instructor Yan

* Create read\_course\_instructor.py, add following queries:

# Django specific settings

import inspect

import os

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'OnlineCourse.settings')

from django.db import connection

# Ensure settings are read

from django.core.wsgi import get\_wsgi\_application

application = get\_wsgi\_application()

from CRUD.models import \*

from datetime import date

# Course has instructors reference field so can be used directly via forward access

courses = Course.objects.filter(instructors\_\_first\_name='Yan')

print("1. Get courses taught by Instructor `Yan`, forward")

print(courses)

print("\n")

# For each instructor, Django creates a implicit course\_set. This is caleld backward access

instructor\_yan = Instructor.objects.get(first\_name='Yan')

print("1. Get courses taught by Instructor `Yan`, backward")

print(instructor\_yan.course\_set.all())

print("\n")

instructors = Instructor.objects.filter(course\_\_name\_\_contains='Cloud')

print("2. Get the instructors of Cloud app dev course")

print(instructors)

print("\n")

courses = Course.objects.filter(instructors\_\_first\_name='Yan')

occupation\_list = set()

for course in courses:

    for learner in course.learners.all():

        occupation\_list.add(learner.occupation)

print("3. Check the occupations of the courses taught by instructor Yan'")

print(occupation\_list)

Above code snippet accesses related objects via both forward and backward access. It also queries related objects with querying parameters to search along the relationship such as from Course to Instructor.

* Run the queries in terminal and check results

python read\_course\_instructors.py

**Query Results**

1. Get courses taught by Instructor `Yan`, forward

<QuerySet [<Course: Name: Cloud Application Development with Database,Description: Develop and deploy application on cloud>]>

1. Get courses taught by Instructor `Yan`, backward

<QuerySet [<Course: Name: Cloud Application Development with Database,Description: Develop and deploy application on cloud>]>

2. Get the instructors of Cloud app dev course

<QuerySet [<Instructor: First name: Yan, Last name: Luo, Is full time: True, Total Learners: 30050>, <Instructor: First name: Joy, Last name: Li, Is full time: False, Total Learners: 10040>]>

3. Check the occupations of the courses taught by instructor Yan'

{'dba', 'data\_scientist', 'developer'}

# Query Related Course, Learner, and User Objects

Create read\_enrollments.py and complete following spanning relationships queries for Course, Learner, and User

1. Get the user information about learner David
2. Get learner David information from user
3. Get all learners for Introduction to Python course
4. Check the occupation list for the courses taught by instructor Yan
5. Check which courses the developer learners are enrolled in Aug, 2020
6. # Django specific settings
7. import inspect
8. import os
9. os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'OnlineCourse.settings')
10. from django.db import connection
11. # Ensure settings are read
12. from django.core.wsgi import get\_wsgi\_application
13. application = get\_wsgi\_application()
14. from CRUD.models import \*
15. from datetime import date

learner\_david = Learner.objects.get(first\_name="David")

print("1. Get the user information about learner `David`")

print(learner\_david.user\_ptr)

user\_david = User.objects.get(first\_name="David")

print("2. Get learner `David` information from user")

print(user\_david.learner)

course = Course.objects.get(name='Introduction to Python')

learners = course.learners.all()

print("3. Get all learners for `Introduction to Python` course")

print(learners)

courses = Course.objects.filter(instructors\_\_first\_name='Yan')

occupation\_list = set()

for course in courses:

    for learner in course.learners.all():

        occupation\_list.add(learner.occupation)

print("4. Check the occupation list for the courses taught by instructor `Yan`")

print(occupation\_list)

enrollments = Enrollment.objects.filter(date\_enrolled\_\_month=8,

                                            date\_enrolled\_\_year=2020,

                                            learner\_\_occupation='developer')

courses\_for\_developers = set()

for enrollment in enrollments:

    course = enrollment.course

    courses\_for\_developers.add(course.name)

print("5. Check which courses developers are enrolled in Aug, 2020")

print(courses\_for\_developers)

* Run the queries in terminal and check results

python read\_enrollments.py

**Query Results**

1. Get the user information about learner `David`

David Smith

2. Get learner `David` information from user

First name: David, Last name: Smith, Date of Birth: 1983-07-16, Occupation: developer, Social Link: https://www.linkedin.com/david/

3. Get all learners for `Introduction to Python` course

<QuerySet [<Learner: First name: Robert, Last name: Lee, Date of Birth: 1999-01-02, Occupation: student, Social Link: https://www.facebook.com/robert/>]>

4. Check the occupation list for the courses taught by instructor `Yan`

{'data\_scientist', 'developer', 'dba'}

5. Check which courses developers are enrolled in Aug, 2020

{'Cloud Application Development with Database'}

# Summary

In this lab, you learned and practiced creating Django Models with relationships and saving those Django Models objects into database, and querying them with filters.

you have learned how to query and access related objects spanning relationships via explicit forward access and implicit backward access

## Topic Summary

At this point in the course, you know:

* The Object-Oriented Programming (OOP) paradigm and the SQL paradigm model data differently.
* Object Relational Mapping, or ORM, bridges the gap between OOP and SQL.
* ORM libraries or tools can map and transfer data stored in a relational database as rows into objects or objects into rows.​
* ORM allows developers to use OOP to query and manipulate data because it transfers objects into rows and rows into objects.
* Django ORM is a Python ORM component that belongs to the Django web application framework.
* Django ORM can help speed up development with databases because for each Django model you define maps to a database table.
* Each Django field maps to a column type.
* Django automatically creates tables once models and fields are defined.
* You can use Django APIs to perform Create, Read, Update, and Delete (CRUD) operations on objects in databases.
* In a Django model, you create an object and call the model’s save method to insert it into the database as a record.
* To read objects using Django Model API, you need to construct a QuerySet using a Manager on model class.
* There are several ways to update database records in Django by updating objects.
* To delete records in a database, you call Django ORM’s Delete method on a model object or query set.