IBM Developer
SKILLS NETWORK

Query Spanning Relationships

Estimated time needed: 15 minutes

Learning Objectives

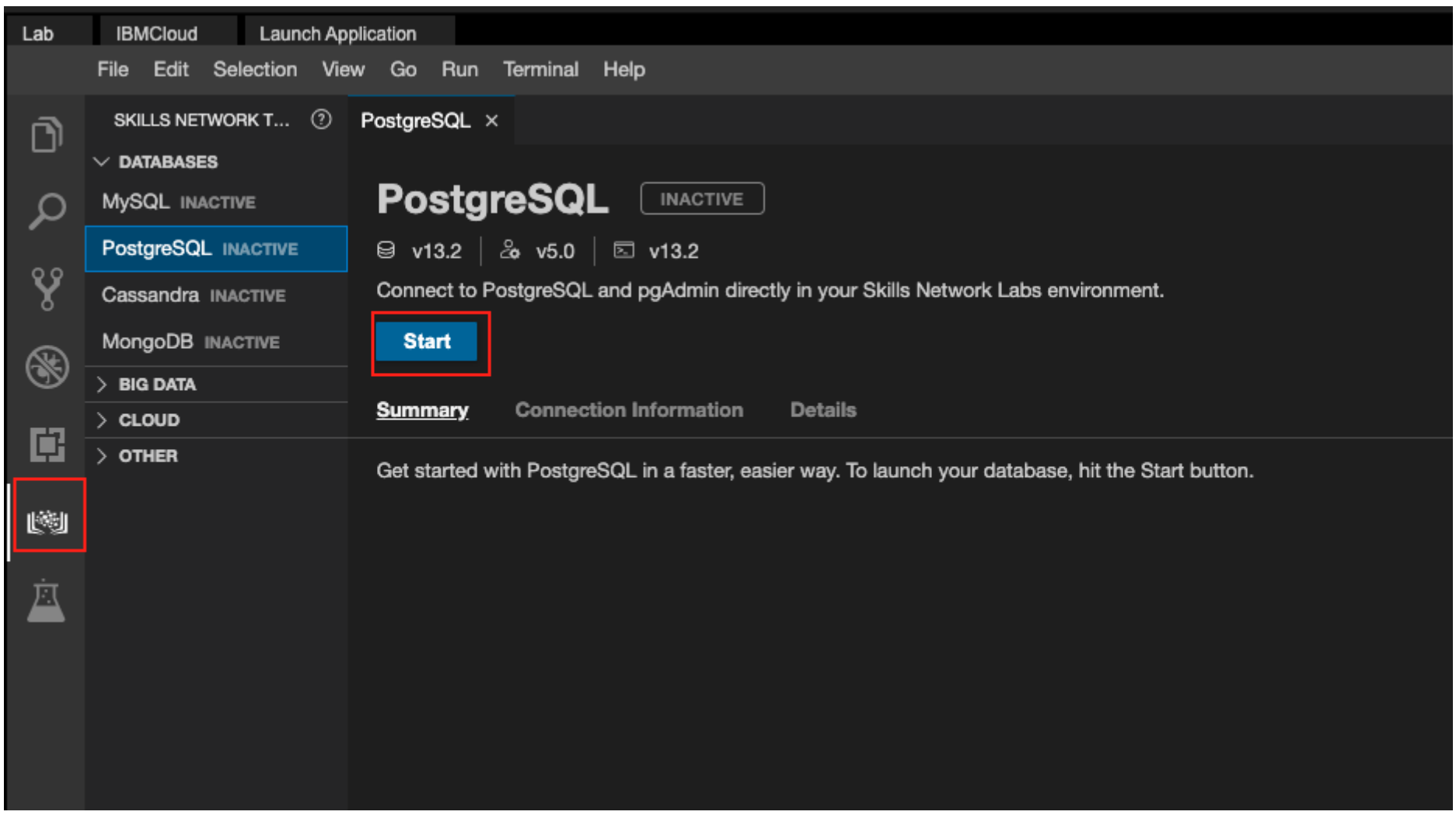
- Query related objects spanning relationships

Start PostgreSQL in Theia

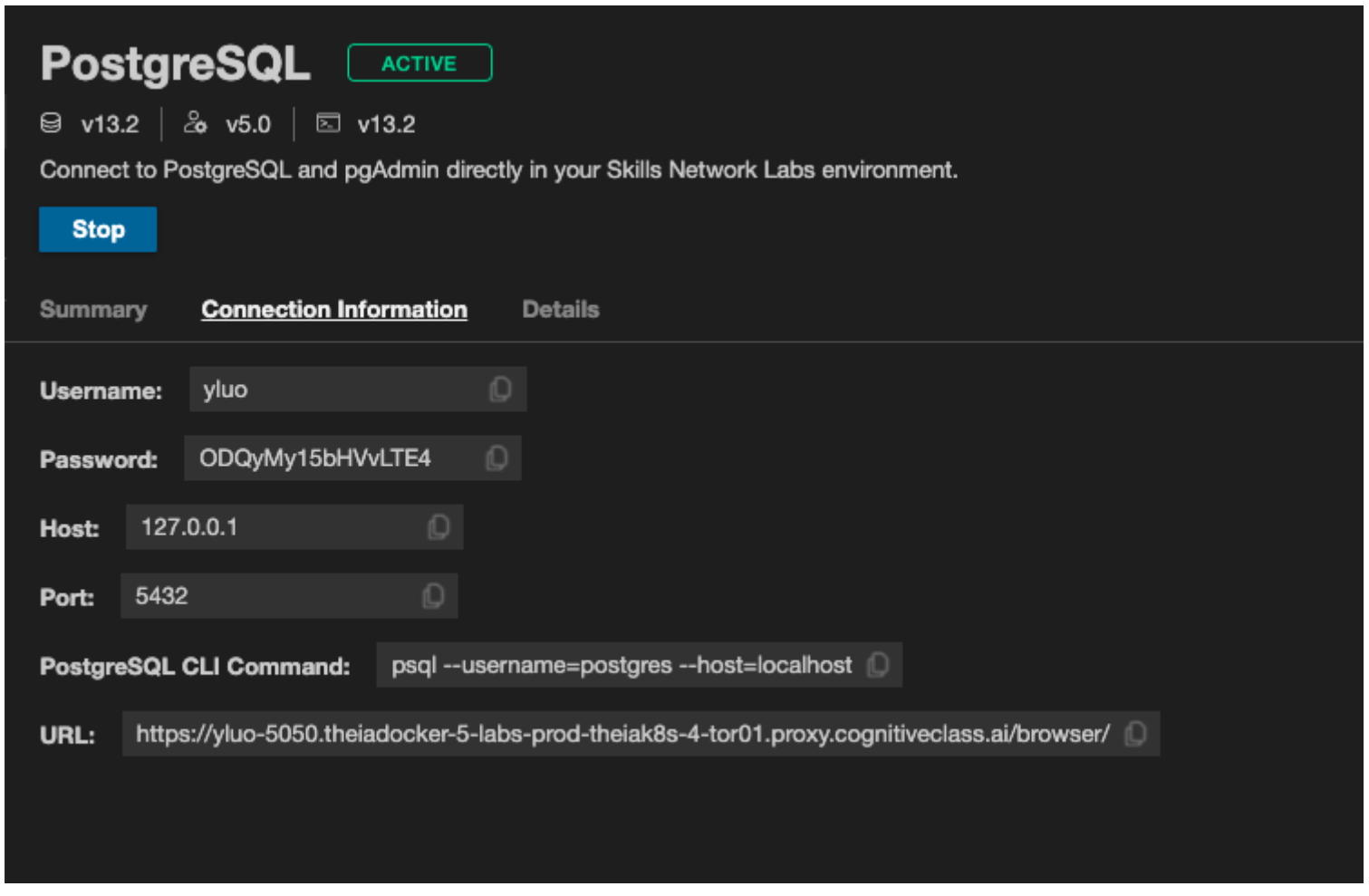
`PostgreSQL`, also known as `Postgres`, is an open-source relational database management system and it is one of the main databases Django uses.

If you are using the Theia environment hosted by Skills Network Labs, a pre-installed PostgreSQL instance is provided for you.

You can start PostgreSQL from UI by finding the SkillsNetwork icon on the left menu bar and selecting PostgreSQL from the DATABASES menu item:



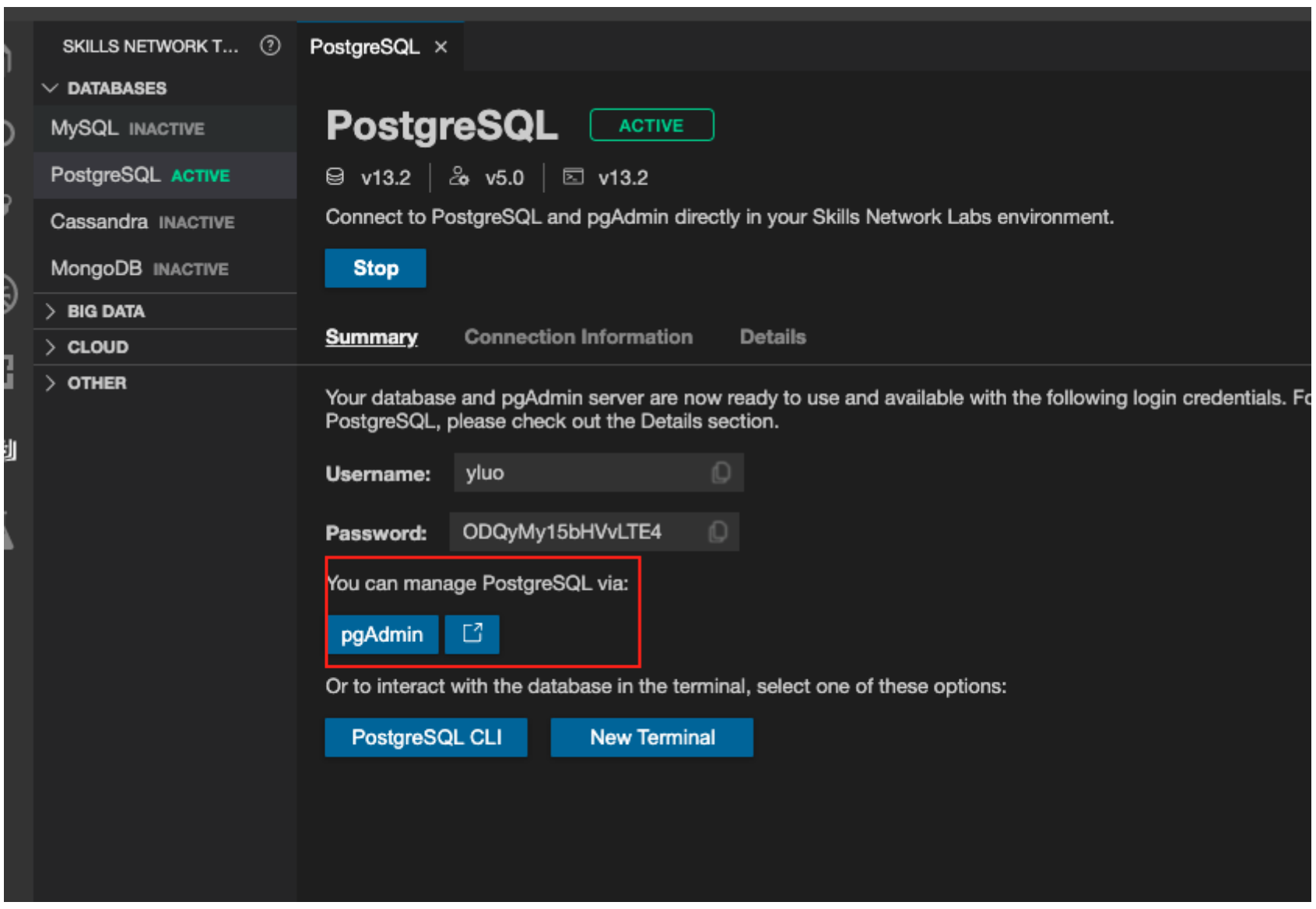
Once the PostgreSQL has been started, you can check the server connection information from the UI. Please markdown the connection information such as generated `username`, `password`, and `host`, etc, which will be used to configure Django app to connect to this database.



- Django needs an adapter called `Psycopg` as an interface to work with PostgreSQL, you can install it using following command:

```
python3 -m pip install psycopg2-binary
```

Also, a `pgAdmin` instance is installed and started for you. It is a popular PostgreSQL admin and development tool for you to manage your PostgreSQL server interactively.



Import a Django ORM Project With Predefined Models

Before starting the lab, make sure your current Theia directory is `/home/project`.

First we need to install Django related packages.

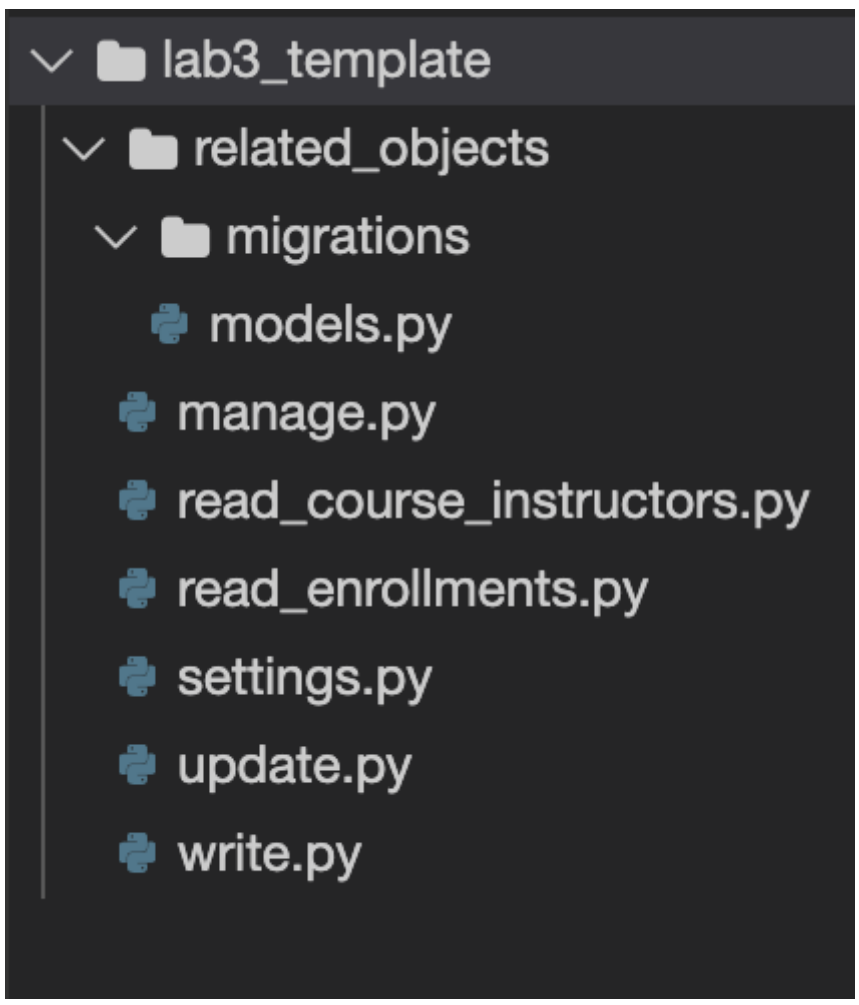
- If the terminal was not open, go to `Terminal > New Terminal` and run:

```
python3 -m pip install Django
```

- Run the following command-lines to download a code template for Lab 3

```
wget "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-CD0251EN-SkillsNetwork/labs/m3_django_orm/lab3_template.zip"
unzip lab3_template.zip
rm lab3_template.zip
```

You Django project should look like following:



- Open `settings.py` and find `DATABASES` section, and replace the values of `USER` and `PASSWORD` to be the generated `PostgreSQL` user and password.

Your `settings.py` file now should look like the following:

```
# PostgreSQL
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD': 'NDg3LXlsdW8tMzA2',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

INSTALLED_APPS = (
    'related_objects',
)

SECRET_KEY = 'SECRET KEY for this Django Project'
```

Activate Models for an `onlinecourse` App

- Open `related_objects/models.py`, you could find the same models for an online course app you created in lab `CRUD on Django Model Objects`.

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`

Now let's activate those models by running migrations

- If your current working directory is not `/home/project/lab3_template`, `cd` to the project folder

```
cd lab3_template
```

- Then generate migration scripts for app `related_objects`

```
python3 manage.py makemigrations related_objects
```

and you should see Django is about to create following tables.

```
Migrations for 'related_objects':
  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
```

- then run migration

```
python3 manage.py migrate
```

and you should see the migration script `related_objects.0001_initial` was executed.

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

Populating Data with Relationships

At this point, you have migrated all the models. Let's try to populate objects with relationships and save them into database.

The `lab3_template/write.py` script first creates the same model objects in lab `CRUD on Django Model Objects`. In addition to that, it adds two `populate_course_instructor_relationships()` and `populate_course_enrollment_relationships()` methods to create relationships by updating the reference fields with objects.

- Now run

```
python3 write.py
```

You should see objects and relationships saved messages in terminal.

```
Course objects saved...
Instructors objects saved...
Learners objects saved...
Lessons objects saved...
Course-instructor relationships saved...
Course-learner relationships saved...
```

#Coding practice: Create More Objects and Relationships Please review and follow the examples in `write.py` to create more objects with relationships.

Making querying span relationships

After data have been populated, we can start query them.

In this lab, 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`
- Open `read_course_instructor.py`, add following queries:

```
# 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 and check results

```
python3 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'}
```

Coding practice: Query Related `Course`, `Learner`, and `User` Objects

Open `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

```
print("1. Get the user information about learner `David`")
learner_david = Learner.objects.get(first_name="David")
print( #<HINT> use the usr_ptr field created by Django for the Inheritance relationship# )

print("2. Get learner `David` information from user")
user_david = User.objects.get(first_name="David")
print( #<HINT> use the learner field created by Django# )


print("3. Get all learners for `Introduction to Python` course")
course = Course.objects.get(name='Introduction to Python')
learners = #<HINT> use the learners field in Course model#
print(learners)

print("4. Check the occupation list for the courses taught by instructor `Yan`")
courses = Course.objects.filter( #<HINT> query the first name of instructor# )
occupation_list = set()
for course in courses:
    for learner in #<HINT>use the learners field in Course Model#  :
        occupation_list.add(learner.occupation)
print(occupation_list)


print("5. Check which courses developers are enrolled in Aug, 2020")
enrollments = Enrollment.objects.filter(date_enrolled__month=8,
                                        date_enrolled__year=2020,
                                        #<HINT>use the occupation field from learner #)
courses_for_developers = set()
for enrollment in enrollments:
    course = enrollment.course
    courses_for_developers.add(course.name)
print(courses_for_developers)
```

▶ Click here to see solution

Summary

In this lab, you have learned how to query and access related objects spanning relationships via explicit forward access and implicit backward access.

Author(s)

[Yan Luo](linkedin.com/in/yan-luo-96288783)

Changelog

Date	Version	Changed by	Change Description
30-Nov-2020	1.0	Yan Luo	Initial version created

© IBM Corporation 2020. All rights reserved.