

Query Spanning Relationships



Estimated time needed: 15 minutes

In this lab, you will learn to query and access related objects spanning relationships.

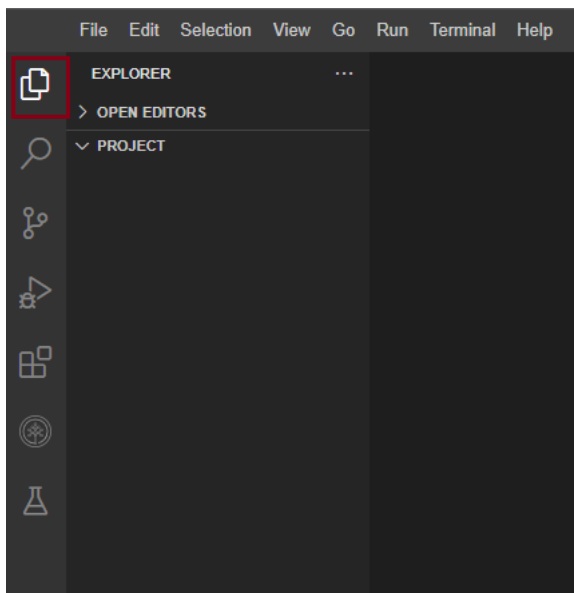
Learning Objectives

- Query related objects spanning relationships

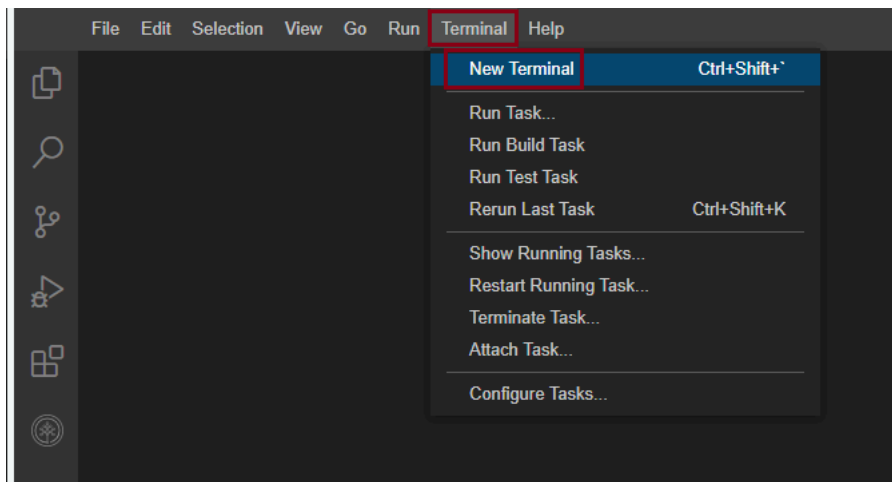
Working with files in Cloud IDE

If you are new to Cloud IDE, this section will show you how to create and edit files, which are part of your project, in Cloud IDE.

To view your files and directories inside Cloud IDE, click on this files icon to reveal it.



Click on New, and then New Terminal.



This will open a new terminal where you can run your commands.

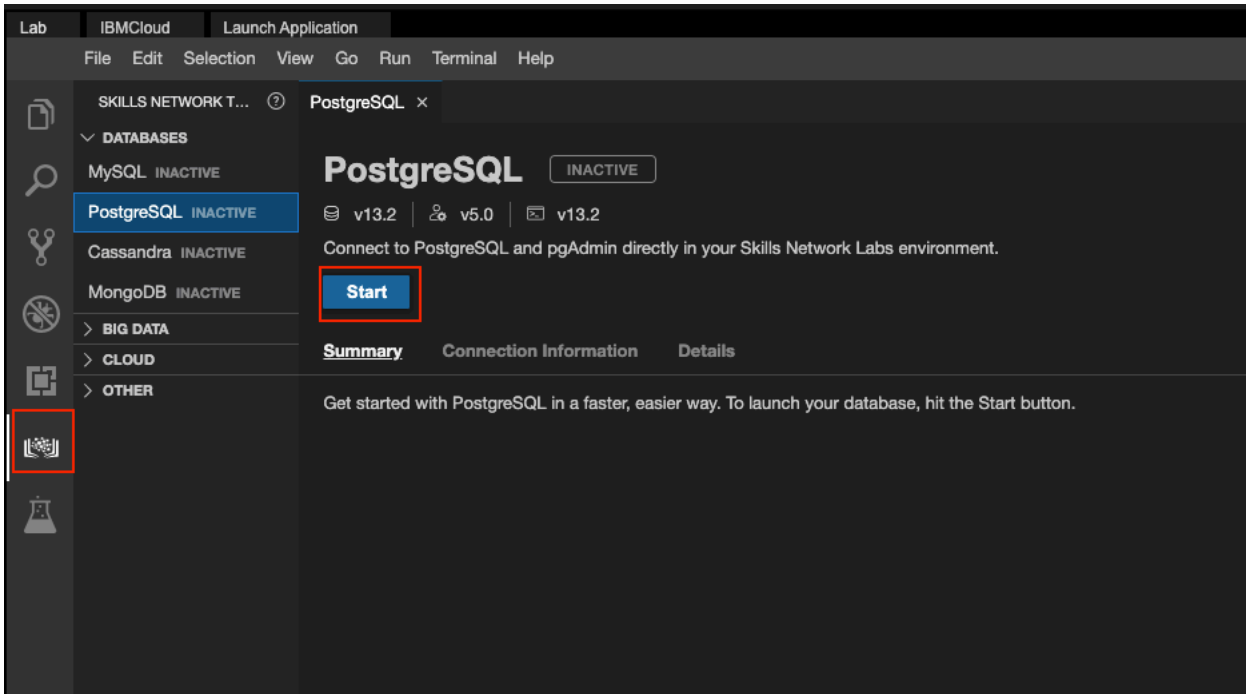
```
Problems theia@theiadocke /home/project x
theia@theiadocke :/home/project$
```

Concepts covered in the lab

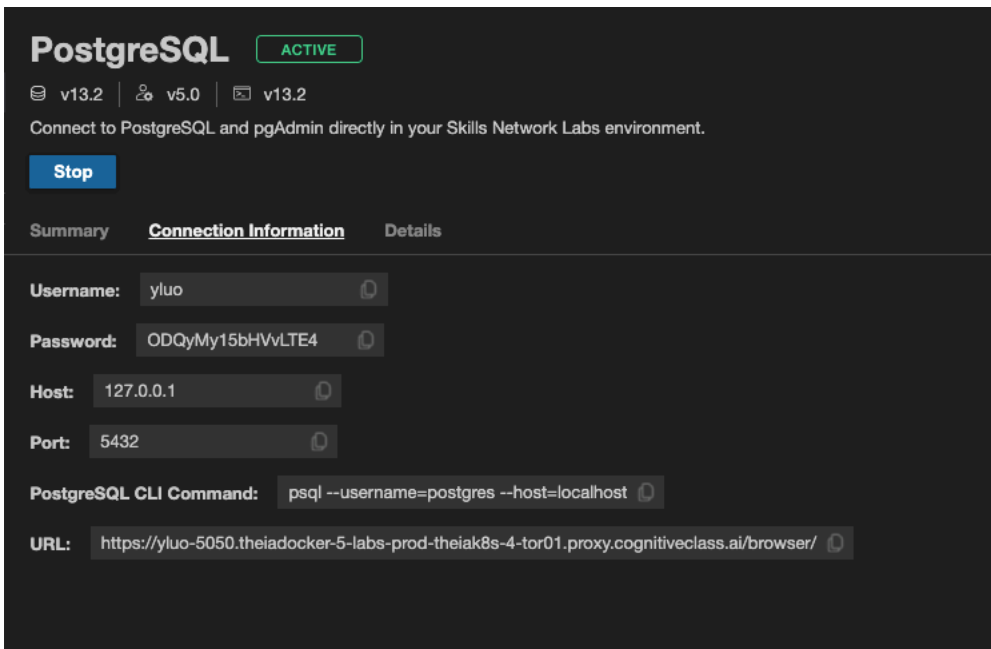
- 1. `QuerySet`: Represents a collection of records in a database, and is required to read objects using Django Model API.
- 2. `get()` method: Returns a single object that matches the lookup criterion.
- 3. `all()` method: Returns a `QuerySet` of all the objects in the database.
- 4. `filter()` method: Returns only the rows that match the search term. It can have lookup parameters such as **greater than**, **less than**, **contains**, or **is null**.
- 5. Forward relationship: If a model has a Foreign key, instances of that model will have access to the related (foreign) object via an attribute of the model.
- 6. Backward relationship: If a model has a Foreign key, instances of the foreign-key model will have access to a Manager that returns all instances of the first model.

Start PostgreSQL in Theia

- If you are using the Theia environment hosted by [Skills Network Labs](#), you can start the pre-installed PostgreSQL instance by finding the SkillsNetwork icon on the left menu bar and selecting PostgreSQL from the DATABASES menu item:



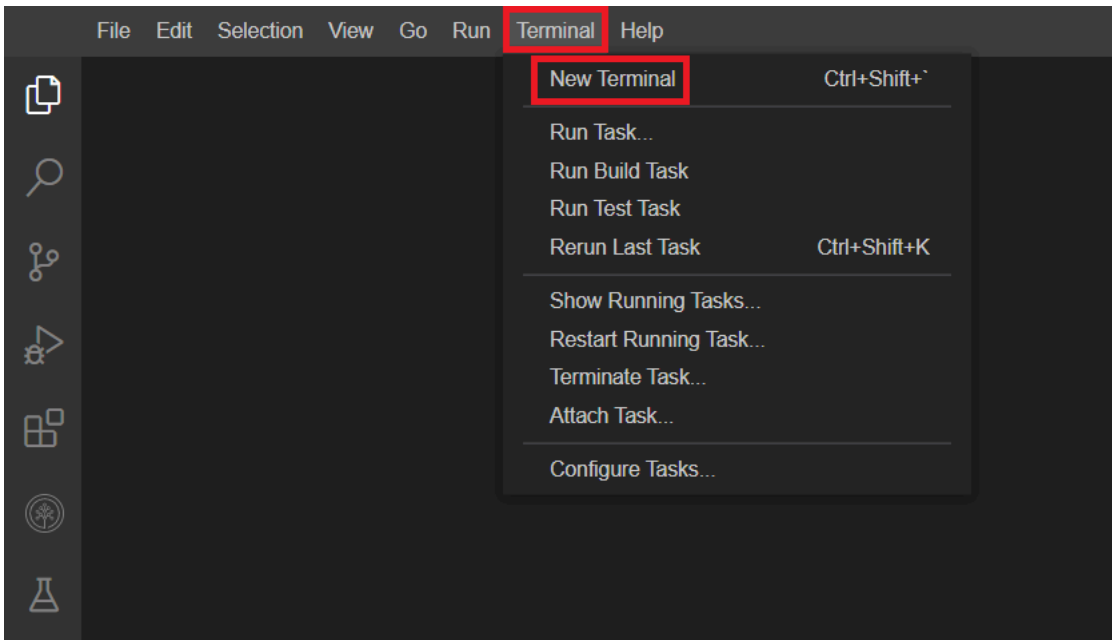
- Once the PostgreSQL has been started, you can check the server connection information from the UI. Please markdown the connection information such as generated username, password, and host, etc, which will be used to configure Django app to connect to this database.



Optional: Start PostgreSQL from terminal

You may also start PostgreSQL from terminal (*if you have not already started it from the UI*):

- If the terminal was not open, go to Terminal > New Terminal



- and run:

```
1. 1
1. start_postgres
```

Copied!

- Once PostgreSQL is started, you can check the server connection information in the terminal. You need to save the connection information such as generated username, password, and host, etc, which will be used to configure Django app to connect to this database.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10

1. Starting your Postgres database....
```

```

2. This process can take up to a minute.
3.
4. Postgres database started, waiting for all services to be ready....
5.
6. Your Postgres database is now ready to use and available with username: postgres password: Nzg3Mi15bHVvLTIz
7.
8. You can access your Postgres database via:
9.  ^ https://yluo-5050.theiadocker-1.proxy.cognitiveclass.ai
10.  ^ql --username=postgres --host=localhost

```

Copied!

- Install these must-have packages before you setup the environment to access postgres.

```

1. 1
2. 2

1. pip install --upgrade distro-info
2. pip3 install --upgrade pip==23.2.1

```

Copied!

Executed!

- You will also see a pgAdmin instance installed and started.

The screenshot shows the PostgreSQL management interface. On the left, a sidebar lists various databases: MySQL (INACTIVE), PostgreSQL (ACTIVE), Cassandra (INACTIVE), MongoDB (INACTIVE), BIG DATA, CLOUD, and OTHER. The main panel is titled 'PostgreSQL' and has an 'ACTIVE' status. It includes version information (v13.2, v5.0, v13.2) and a 'Stop' button. Below this, there's a 'Summary' tab selected, showing login credentials: Username: yluo and Password: ODQyMy15bHVvLTE4. A red box highlights the text 'You can manage PostgreSQL via:' with two buttons: 'pgAdmin' and a link icon. At the bottom, there are buttons for 'PostgreSQL CLI' and 'New Terminal'.

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 make sure your current Theia directory is `/home/project`.

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

```

1. 1
2. 2
3. 3

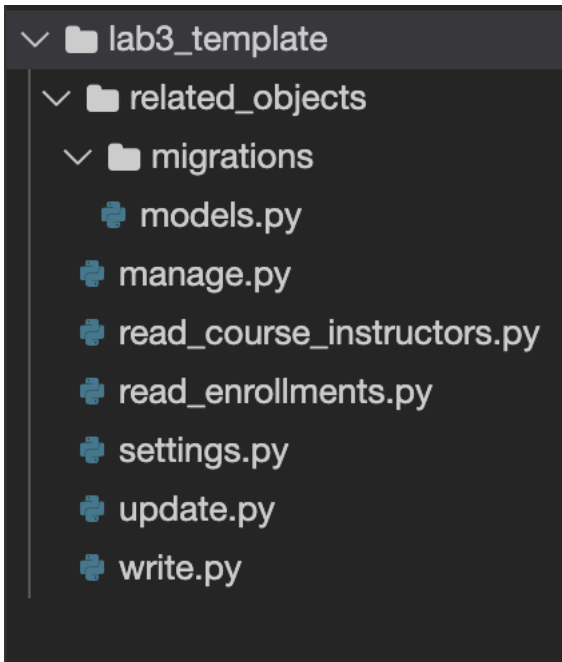
1. wget "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-CD0251EN-SkillsNetwork/labs/m3_django_orm/lab3_template.zip"
2. unzip lab3_template.zip
3. rm lab3_template.zip

```

Copied!

Executed!

Your Django project should look like following:



- Open settings.py and find the DATABASES section. Replace the value of the PASSWORD, to be the generated PostgreSQL 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

```
1. 1
1. cd lab3_template
```

Copied!

Let's set up a virtual environment which has all the packages we need.

```
1. 1
2. 2
3. 3
4. 4

1. pip install virtualenv
2. virtualenv djangoenv
3. source djangoenv/bin/activate
4. pip install Django psycopg2-binary
```

Copied!

Executed!

- Then generate migration scripts for app `related_objects`

```
1. 1
1. python3 manage.py makemigrations related_objects
```

Copied!

Executed!

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

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10

1. Migrations for 'related_objects':
2.   crud/migrations/0001_initial.py
3.   - Create model Course
4.   - Create model User
5.   - Create model Instructor
6.   - Create model Learner
7.   - Create model Lesson
8.   - Create model Enrollment
9.   - Add field instructors to course
10.  - Add field learners to course
```

Copied!

- then run migration

```
1. 1
1. python3 manage.py migrate
```

Copied!

Executed!

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

```
1. 1
2. 2
3. 3
4. 4

1. Operations to perform:
2.   Apply all migrations: related_objects
3. Running migrations:
4.   Applying related_objects.0001_initial... OK
```

Copied!

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

```
1. 1
```

```
1. python3 write.py
```

Copied! Executed!

You should see objects and relationships saved messages in terminal.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
```

```
1. Course objects saved...
2. Instructors objects saved...
3. Learners objects saved...
4. Lessons objects saved...
5. Course-instructor relationships saved...
6. Course-learner relationships saved...
```

Copied!

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:
 - Get courses taught by Instructor Yan, via both forward (explicit) and backward (implicit) access
 - Get the instructors of Cloud app dev course
 - Check the occupations of the courses taught by instructor Yan
- Open `read_course_instructor.py`, add following queries:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
```

```
1. # Course has instructors reference field so can be used directly via forward access
2. courses = Course.objects.filter(instructors__first_name='Yan')
3. print("1. Get courses taught by Instructor `Yan`, forward")
4. print(courses)
5.
6. print("\n")
7. # For each instructor, Django creates a implicit course_set. This is called backward access
8. instructor_yan = Instructor.objects.get(first_name='Yan')
9. print("1. Get courses taught by Instructor `Yan`, backward")
10. print(instructor_yan.course_set.all())
11.
12. print("\n")
13. instructors = Instructor.objects.filter(course__name__contains='Cloud')
14. print("2. Get the instructors of Cloud app dev course")
15. print(instructors)
16.
17. print("\n")
18. courses = Course.objects.filter(instructors__first_name='Yan')
19. occupation_list = set()
20. for course in courses:
21.     for learner in course.learners.all():
22.         occupation_list.add(learner.occupation)
```

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

Copied!

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

1. 1

1. python3 read_course_instructors.py

Copied!

- Query results:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11

1. 1. Get courses taught by Instructor `Yan`, forward
2. <QuerySet [<Course: Name: Cloud Application Development with Database,Description: Develop and deploy application on cloud>]>
3.
4. 1. Get courses taught by Instructor `Yan`, backward
5. <QuerySet [<Course: Name: Cloud Application Development with Database,Description: Develop and deploy application on cloud>]>
6.
7. 2. Get the instructors of Cloud app dev course
8. <QuerySet [<Instructor: First name: Yan, Last name: Luo, Is full time: True, Total Learners: 30050>, <Instructor: First name: Joy, Last name: Li, Is
9.
10. 3. Check the occupations of the courses taught by instructor Yan'
11. {'dba', 'data_scientist', 'developer'}
```

Copied!

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

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30

1. print("1. Get the user information about learner `David`")
2. learner_david = Learner.objects.get(first_name="David")
3. print( #<HINT> use the usr_ptr field created by Django for the Inheritance relationship# )
4.
5. print("2. Get learner `David` information from user")
6. user_david = User.objects.get(first_name="David")
```



```
7. print( #<HINT> use the learner field created by Django# )
8.
9. print("3. Get all learners for `Introduction to Python` course")
10. course = Course.objects.get(name='Introduction to Python')
11. learners = #<HINT> use the learners field in Course model#
12. print(learners)
13.
14. print("4. Check the occupation list for the courses taught by instructor `Yan`")
15. courses = Course.objects.filter( #<HINT> query the first name of instructor# )
16. occupation_list = set()
17. for course in courses:
18.     for learner in #<HINT>use the learners field in Course Model# :
19.         occupation_list.add(learner.occupation)
20. print(occupation_list)
21.
22. print("5. Check which courses developers are enrolled in Aug, 2020")
23. enrollments = Enrollment.objects.filter(date_enrolled__month=8,
24.                                         date_enrolled__year=2020,
25.                                         #<HINT>use the occupation field from learner #)
26. courses_for_developers = set()
27. for enrollment in enrollments:
28.     course = enrollment.course
29.     courses_for_developers.add(course.name)
30. print(courses_for_developers)
```

Copied!

► [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

© IBM Corporation. All rights reserved.