# Lab: Django Models Relations

This document defines the problems for the in-class lab for the [**Python ORM course @ Software University**](https://softuni.bg/modules/137/python-db).

Submit your solutions to the SoftUni [**Judge system**](https://judge.softuni.org/Contests/4327/Django-Models-Relations-Lab).

For this lab, you are given an empty **ORM project skeleton** (you can download it from the current lesson's resources). Your task for today is to **create a University Management System** containing the models: **"Student"**, **"Lecturer"**, "**Subject"**, **"StudentEnrollment"** and **"LecturerProfile"**.

## The Lecturer

In the **main\_app** create two models called **"Lecturer"** and **"Subject"**:

* **Lecturer**
  + **first\_name**
    - A **character** field.
    - It has a **maximum length** of **100** characters.
    - Represents the first name of the lecturer.
  + **last\_name**
    - A **character** field.
    - It has a **maximum length** of **100** characters.
    - Represents the last name of the lecturer.
* **Subject**
  + **name**
    - A **character** field.
    - It has a **maximum length** of **100** characters.
    - Represents the name of the subject.
  + **code**
    - A **character** field.
    - It has a **maximum length** of **10** characters.
    - Represents the code of the subject.
  + **lecturer**
    - A **foreign key** field.
    - Establishes a **many-to-one relationship** with the **"Lecturer" model**, associating each subject with a lecturer.
    - When a **lecturer is deleted**, the **"lecturer" field in all subjects** **that reference the deleted lecturer is set to NULL**.
* **Hint**: Implement a **\_\_str\_\_** method for the **Lecturer Model** that **returns** the **first** and **last names**, according to the examples given below.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import Lecturer, Subject  lecturer1 = Lecturer.objects.create(first\_name="John", last\_name="Doe")  lecturer2 = Lecturer.objects.create(first\_name="Jane", last\_name="Smith")  Subject.objects.create(name="Mathematics", code="MATH101", lecturer=lecturer1)  Subject.objects.create(name="History", code="HIST101", lecturer=lecturer2)  Subject.objects.create(name="Physics", code="PHYS101", lecturer=lecturer1)  math\_subject = Subject.objects.get(name="Mathematics")  math\_lecturer = math\_subject.lecturer  print(f"The lecturer for Mathematics is {math\_lecturer}.")  history\_subject = Subject.objects.get(name="History")  history\_lecturer = history\_subject.lecturer  print(f"The lecturer for History is {history\_lecturer}.")  physics\_subject = Subject.objects.get(name="Physics")  physics\_lecturer = physics\_subject.lecturer  print(f"The lecturer for Physics is {physics\_lecturer}.") |
| **Output** |
| The lecturer for Mathematics is John Doe.  The lecturer for History is Jane Smith.  The lecturer for Physics is John Doe. |

## The Student

In the **main\_app** create one **additional** model called **"Student"**:

* **Student**
  + **student\_id**
    - A **character** field.
    - It has a **maximum length** of **10** characters.
    - It should be set as a **primary key**.
    - Represents the identification student number of the student.
  + **first\_name**
    - A **character** field.
    - It has a **maximum length** of **100** characters.
    - Represents the first name of the student.
  + **last\_name**
    - A **character** field.
    - It has a **maximum length** of **100** characters.
    - Represents the last name of the student.
  + **birth\_date**
    - A **date** field.
    - Represents the birth date of the student.
  + **email**
    - An **email** field.
    - Each email should be **unique**.
    - Represents the email of the student.
  + **subjects**
    - A **many-to-many** field.
    - Establishes a **many-to-many relationship** with the **"Subject" model**, allowing each student to be associated with one or more subjects.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import Subject, Student  ***# Keep the data from the previous exercise, so you can reuse it***  student1 = Student.objects.create(student\_id="M1051", first\_name="Alice", last\_name="Johnson", birth\_date="2000-01-15", email="a.johnson@abv.bg")  student2 = Student.objects.create(student\_id="S217", first\_name="Bob", last\_name="Smith", birth\_date="2001-05-20", email="bobby@gmail.com")  subject1 = Subject.objects.get(name="Mathematics")  subject2 = Subject.objects.get(name="History")  subject3 = Subject.objects.get(name="Physics")  student1.subjects.add(subject1, subject2)  student2.subjects.add(subject1, subject2, subject3)  math\_subject = Subject.objects.get(name="Mathematics")  math\_students = math\_subject.student\_set.all()  for student in math\_students:  print(f"{student.first\_name} {student.last\_name} is enrolled in Mathematics.")  history\_subject = Subject.objects.get(name="History")  history\_students = history\_subject.student\_set.all()  for student in history\_students:  print(f"{student.first\_name} {student.last\_name} is enrolled in History.")  physics\_subject = Subject.objects.get(name="Physics")  physics\_students = physics\_subject.student\_set.all()  for student in physics\_students:  print(f"{student.first\_name} {student.last\_name} is enrolled in Physics.") |
| **Output** |
| Alice Johnson is enrolled in Mathematics.  Bob Smith is enrolled in Mathematics.  Alice Johnson is enrolled in History.  Bob Smith is enrolled in History.  Bob Smith is enrolled in Physics. |

## The Enrollment

We want to improve our management system, so we want you to add a **"through"** option to the **Student's** "**subjects"** field and a **"through"** table:

* **StudentEnrollment**
  + **student**
    - A **foreign key** field.
    - Establishes a **many-to-one relationship** with the **"Student" model**, associating each subject with a student.
    - When a **student is deleted**, **all rows** that reference the student **should be deleted**.
  + **subject**
    - A **foreign key** field.
    - Establishes a **many-to-one relationship** with the **"Subject" model**, associating each student with a subject.
    - When a **subject is deleted**, **all rows** that reference the subject **should be deleted**.
  + **enrollment\_date**
    - A **date** field.
    - When **no date is given**, it should be **set to the current date (today) when the object is first created**.
    - Represent the student's enrollment date for the subject.
  + **grade**
    - A **character** field.
    - It has a **maximum length** of **1** character.
    - Holds **predefined choices** of each grade: **"A"**, **"B"**, **"C"**, **"D"**, and **"F"**.
    - Represents the final grade of the student for the subject.
* **Hint**: Implement a **\_\_str\_\_** method for the **Subject Model** that **returns** the **subject's name**, according to the examples given below.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import Student  ***# Keep the data from the previous exercises, so you can reuse it***  student = Student.objects.get(student\_id="S217")  student\_enrollments = student.studentenrollment\_set.all()  for enrollment in student\_enrollments:  print(f"{student.first\_name} {student.last\_name} is enrolled in {enrollment.subject}.") |
| **Output** |
| Bob Smith is enrolled in Mathematics.  Bob Smith is enrolled in History.  Bob Smith is enrolled in Physics. |

### Hint

When we **try to add the "through" table to the database** when there is already applied many-to-many relation in Django, we are going to hit a **ValueError** with the message "Cannot alter field main\_app.Student.subjects into main\_app.Student.subjects - they are not compatible types (you cannot alter to or from M2M fields, or add or remove through= on M2M fields)". This is because **Django has already created a "through" table when we first applied the many-to-many relation**.

Documentation on the following content can be found here: <https://docs.djangoproject.com/en/5.0/howto/writing-migrations/#changing-a-manytomanyfield-to-use-a-through-model>.

To **avoid the error**, we need to **make some changes to the last created migration file**:

|  |
| --- |
| **0003\_studentenrollment\_alter\_student\_subjects.py** |
| **from django.db import migrations, models**  **import django.db.models.deletion**  **from datetime import date**  **class Migration(migrations.Migration):**  **dependencies = [**  **('main\_app', '0002\_student'),**  **]**  ***# First, add a state operation that creates the table studentenrollment, sets the table name for the StudentEnrollment model to match the existing table name created by the previous migration, and alters the subjects field in the student table.***  **state\_operations = [**  **migrations.CreateModel(**  **name='StudentEnrollment',**  **fields=[**  **('id', models.BigAutoField(auto\_created=True, primary\_key=True, serialize=False, verbose\_name='ID')),**  **('student', models.ForeignKey(on\_delete=django.db.models.deletion.CASCADE, to='main\_app.student')),**  **('subject', models.ForeignKey(on\_delete=django.db.models.deletion.CASCADE, to='main\_app.subject')),**  **],**  **),**  **migrations.AlterModelTable(**  **name='studentenrollment',**  **table='main\_app\_student\_subjects',**  **),**  **migrations.AlterField(**  **model\_name='student',**  **name='subjects',**  **field=models.ManyToManyField(through='main\_app.StudentEnrollment', to='main\_app.Subject'),**  **),**  **]**  ***# Next, in the operations list, we add a specialized operation called SeparateDatabaseAndState which allows us to alter the state and the database separately. In this specialized operation, we will add the above-written state operations.***  **operations = [**  **migrations.SeparateDatabaseAndState(state\_operations=state\_operations),**  ***# We need to add the additional fields from the StudentEnrollment model: "entrollment\_date" and "grade".***  **migrations.AddField(**  **model\_name='StudentEnrollment',**  **name='enrollment\_date',**  **field=models.DateField(default=date.today),**  **),**  **migrations.AddField(**  **model\_name='StudentEnrollment',**  **name='grade',**  **field=models.CharField(blank=True, choices=[('A', 'A'), ('B', 'B'), ('C', 'C'), ('D', 'D'), ('F', 'F')],**  **max\_length=1, null=True),**  **),**  ***# Finally, we reset the name (which is set to main\_app\_student\_subjects above) to use the default naming convention***  **migrations.AlterModelTable(**  **name='studentenrollment',**  **table=None,**  **),**  **]** |

## The Lecturer Profile

In the **main\_app** create one more **additional** model called **"LecturerProfile"**:

* **LecturerProfile**
  + **lecturer**
    - A **one-to-one** field.
    - Establishes a **one-to-one relationship** with the **"Lecturer" model**, associating the profile data with the lecturer.
    - When a **lecturer is deleted**, **the profile** that references the lecturer **should be deleted too**.
  + **email**
    - An **email** field.
    - Each email should be **unique**.
    - Represents the email of the lecturer.
  + **bio**
    - A **text** field.
    - **Optional** filed.
    - Represents the bio of the lecturer.
  + **office\_location**
    - A **character** field.
    - It has a **maximum length** of **100** characters.
    - **Optional** filed.
    - Represents where the office of the lecturer is located (address).

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import Lecturer, LecturerProfile  ***# Keep the data from the previous exercises, so you can reuse it***  lecturer = Lecturer.objects.get(first\_name='John', last\_name="Doe")  lecturer\_profile = LecturerProfile.objects.create(lecturer=lecturer, email="john.doe@university.lecturers.com", bio="A skilled and passionate math lecturer", office\_location="Sofia, Al. Stamobolyiski Str, Faculty of Mathematics and Computer Science, Room 101")  lecturer\_profile\_from\_db = LecturerProfile.objects.get(email='john.doe@university.lecturers.com')  print(f"{lecturer\_profile\_from\_db.lecturer.first\_name} {lecturer\_profile\_from\_db.lecturer.last\_name} has a profile.") |
| **Output** |
| John Doe has a profile. |