# Exercises: Django Models Relations

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

Submit your solutions in the SoftUni [**Judge system**](https://judge.softuni.org/Contests/4328/Django-Models-Relations-Exercise).

## Library

Write a Django model called **"Author"** with the provided **field**:

* **name** - character field, **consisting of a maximum of 40 characters.**

Write a **second** Django model called **"Book"** with the provided **fields**:

* **title** - character field, **consisting of a maximum of 40 characters.**
* **price** - decimal field, with a **maximum** of **5 digits** and **2 decimal places**.
* **author** - many-to-one relation to the **"Author"** class. If an author is **deleted**, you should **automatically** **delete** all the **related** books.

### Functions inside the caller.py file

**show\_all\_authors\_with\_their\_books()** **returns** a string with **authors** and their **books**, **separated** by **comma and space -** "**,** ", **ordered** **by** the **id** of the **author** (**ascending**) as follows:

* **"{author\_name\_1} has written - {book\_1, book\_2, book\_N}!**

**…**

**{author\_name\_N} has written - {book\_1, book\_2, book\_N}!"**

**Note:** If an **author** does not have any **books**, **continue** to the next one.

**delete\_all\_authors\_without\_books()** **deletes** all **authors** that do not have any **book** related to them.

### Examples

|  |
| --- |
| **Test code - caller.py** |
| # Create authors author1 = Author.objects.create(name="J.K. Rowling") author2 = Author.objects.create(name="George Orwell") author3 = Author.objects.create(name="Harper Lee") author4 = Author.objects.create(name="Mark Twain")  # Create books associated with the authors book1 = Book.objects.create(  title="Harry Potter and the Philosopher's Stone",  price=19.99,  author=author1 ) book2 = Book.objects.create(  title="1984",  price=14.99,  author=author2 )  book3 = Book.objects.create(  title="To Kill a Mockingbird",  price=12.99,  author=author3 )  # Display authors and their books authors\_with\_books = show\_all\_authors\_with\_their\_books() print(authors\_with\_books)  # Delete authors without books delete\_all\_authors\_without\_books() print(Author.objects.count()) |
| **Output** |
| J.K. Rowling has written - Harry Potter and the Philosopher's Stone!  George Orwell has written - 1984!  Harper Lee has written - To Kill a Mockingbird!  3 |

## Music App

Write a Django model called **"Song"** with the provided **fields**:

* **title** - character field, **consisting of a maximum of 100 characters, unique.**

Write a **second** Django model called **"Artist"** with the provided **fields**:

* **name** - character field, **consisting of a maximum of 100 characters, unique.**
* **songs** - many-to-many relation to the **"Song"** class. The field has a **related name** set to "**artists**".

### Functions inside the caller.py file

**add\_song\_to\_artist(artist\_name: str, song\_title: str)** gets the **artist** object by the **artist's name** and the **song** object by the **song's title**, and **adds** the **song** object to the **artist's** **songs** collection.

**get\_songs\_by\_artist(artist\_name: str)** **returns all** the **song** objects from the **songs** collection in a **queryset**, **ordered by song id (descending)** for the given **artist**.

**remove\_song\_from\_artist(artist\_name: str, song\_title: str)** gets the **artist** object by the **artist's name** and the **song** object by the **song's title**, and **removes** the **song** object from the **artist's** **songs** collection.

### Examples

|  |
| --- |
| **Test code - caller.py** |
| # Create artists artist1 = Artist.objects.create(name="Daniel Di Angelo") artist2 = Artist.objects.create(name="Indila") # Create songs song1 = Song.objects.create(title="Lose Face") song2 = Song.objects.create(title="Tourner Dans Le Vide") song3 = Song.objects.create(title="Loyalty")  # Add a song to an artist add\_song\_to\_artist("Daniel Di Angelo", "Lose Face") add\_song\_to\_artist("Daniel Di Angelo", "Loyalty") add\_song\_to\_artist("Indila", "Tourner Dans Le Vide")  # Get all songs by a specific artist songs = get\_songs\_by\_artist("Daniel Di Angelo") for song in songs:  print(f"Daniel Di Angelo: {song.title}")  # Get all songs by a specific artist songs = get\_songs\_by\_artist("Indila") for song in songs:  print(f"Indila: {song.title}")  # Remove a song from an artist remove\_song\_from\_artist("Daniel Di Angelo", "Lose Face")  # Check if the song is removed songs = get\_songs\_by\_artist("Daniel Di Angelo")  for song in songs:  print(f"Songs by Daniel Di Angelo after removal: {song.title}") |
| **Output** |
| Daniel Di Angelo: Loyalty  Daniel Di Angelo: Lose Face  Indila: Tourner Dans Le Vide  Songs by Daniel Di Angelo after removal: Loyalty |

## Shop

Write a Django model called **"Product"** with the provided **field**:

* **name** - character field, **consisting of a maximum of 100 characters, unique.**

Write a **second** Django model called **"Review"** with the provided **fields**:

* **description** - text field, **consisting of a maximum of 200 characters.**
* **rating** – positive small integer field.
* **product** - many-to-one relation to the **"Product"** class. If a product is **deleted**, you should **automatically** **delete** all the **related** reviews. The field has a **related name** set to "**reviews**".

### Functions inside the caller.py file

**calculate\_average\_rating\_for\_product\_by\_name(product\_name: str)** **returns** the calculated **average** **rating** for a given product by its **name**.

**get\_reviews\_with\_high\_ratings(threshold: int)** **returns** all reviews with **greater than** or **equal** ratings to the **threshold**.

**get\_products\_with\_no\_reviews()** **returns** all **products** that do **NOT** **have** any related **reviews, ordered by** product **name** (**descending**).

**delete\_products\_without\_reviews()** **deletes** all the **products** that do not have any **related** **reviews.**

### Examples

|  |
| --- |
| **Test code - caller.py** |
| *# Create some products* product1 = Product.objects.create(name="Laptop") product2 = Product.objects.create(name="Smartphone") product3 = Product.objects.create(name="Headphones") product4 = Product.objects.create(name="PlayStation 5")  *# Create some reviews for products* review1 = Review.objects.create(description="Great laptop!", rating=5, product=product1) review2 = Review.objects.create(description="The laptop is slow!", rating=2, product=product1) review3 = Review.objects.create(description="Awesome smartphone!", rating=5, product=product2)  *# Run the function to get products without reviews* products\_without\_reviews = get\_products\_with\_no\_reviews() print(f"Products without reviews: {', '.join([p.name *for* p *in* products\_without\_reviews])}")  *# Run the function to delete products without reviews* delete\_products\_without\_reviews() print(f"Products left: {Product.objects.count()}")  *# Calculate and print the average rating* print(calculate\_average\_rating\_for\_product\_by\_name("Laptop")) |
| **Output** |
| Products without reviews: PlayStation 5, Headphones  Products left: 2  3.5 |

## License

Write a Django model called **"Driver"** with the provided **fields**:

* **first\_name** - character field, **consisting of a maximum of 50 characters.**
* **last\_name** - character field, **consisting of a maximum of 50 characters.**

Write a **second** Django model called **"DrivingLicense"** with the provided **fields**:

* **license\_number** - character field, **consisting of a maximum of 10 characters, unique.**
* **issue\_date** - date field.
* **driver** - one-to-one relation to the **"Driver"** class. If a driver is **deleted**, you should **automatically** **delete** the **related** license. The field has a **related name** set to "**license**".

### Functions inside the caller.py file

**calculate\_licenses\_expiration\_dates()** **calculates** the **expiration date** for all **licenses**. **The expiration date** is **365** days after the **issue date**. **Return** the **license** **number** and the **expiration date** as a string, **ordered** **by** **license number** **(descending**) as follows:

* **"License with number: {license\_number\_1} expires on {expiration\_date\_1}!**

**…**

**License with number: {license\_number\_N} expires on {expiration\_date\_N}!"**

**get\_drivers\_with\_expired\_licenses(due\_date: date)** **returns** all drivers that have **expired licenses**. A **license** counts as **expired** when the **expiration date** is one or more days after **the due date.**

### Examples

|  |
| --- |
| **Test code - caller.py** |
| # Create drivers driver1 = Driver.objects.create(first\_name="Tanya", last\_name="Petrova") driver2 = Driver.objects.create(first\_name="Ivan", last\_name="Yordanov")  # Create licenses associated with drivers license1 = DrivingLicense.objects.create(license\_number="123", issue\_date=date(2022, 10, 6), driver=driver1)  license2 = DrivingLicense.objects.create(license\_number="456", issue\_date=date(2022, 1, 1), driver=driver2)  # Calculate licenses expiration dates expiration\_dates = calculate\_licenses\_expiration\_dates() print(expiration\_dates)  # Get drivers with expired licenses drivers\_with\_expired\_licenses = get\_drivers\_with\_expired\_licenses(date(2023, 1, 1))  for driver in drivers\_with\_expired\_licenses:  print(f"{driver.first\_name} {driver.last\_name} has to renew their driving license!") |
| **Output** |
| License with number: 456 expires on 2023-01-01!  License with number: 123 expires on 2023-10-06!  Tanya Petrova has to renew their driving license! |

## Car Registration

Write a Django model called **"Owner"** with the provided information:

* **name** - character field, **consisting of a maximum of 50 characters.**

Write a **second** Django model called **"Car"** with the provided information:

* **model** - character field, **consisting of a maximum of 50 characters.**
* **year** - positive integer field.
* **owner** - many-to-one relation to the **"Owner"** class. If an owner is **deleted**, you should **automatically** **delete** all the **related** cars. The field is **optional** and has a **related name** set to "**cars**".

Write a **third** Django model called **"Registration"** with the provided information:

* **registration\_number** - character field, **consisting of a maximum of 10 characters, unique.**
* **registration\_date** - date field, **optional**.
* **car** - one-to-one relation to the **"Car"** class. If a car is **deleted**, you should **automatically** **delete** the **related** registration. The field is **optional** and has a **related name** set to "**registration**".

### Functions inside the caller.py file

**register\_car\_by\_owner(owner: Owner)** that **register cars with the given owner object**:

* First, get the first **registration** that is not **related** to any **car**. After that get the **first** car without **registration**.
* Then, you should **set** the new **registration** to the **car object**. The **registration date** should be set to the current **day** of the **registration** (**today**) to the **registration object**. **Do not forget to save all the changes in the database.**
* Lastly, **return** thecar **model,** the **owner's name,** and the **registration number** as a string as follows:

**"Successfully registered {car\_model} to {owner\_name} with registration number {registration\_number}."**

### Examples

|  |
| --- |
| **Test code - caller.py** |
| # Create owners  owner1 = Owner.objects.create(name='Ivelin Milchev') owner2 = Owner.objects.create(name='Alice Smith')  # Create cars car1 = Car.objects.create(model='Citroen C5', year=2004) car2 = Car.objects.create(model='Honda Civic', year=2021)  # Create instances of the Registration model for the cars registration1 = Registration.objects.create(registration\_number='TX0044XA') registration2 = Registration.objects.create(registration\_number='XYZ789')  print(register\_car\_by\_owner(owner1)) |
| **Output** |
| Successfully registered Citroen C5 to Ivelin Milchev with registration number TX0044XA. |

### Constraints

* The **owner** will always have at least **one** **car**.

## \*Car Admin Setup

You are going to work with the model "**Car**" from **exercise 5**. In the "**admin.py**" file inside the "**main\_app**" create a **super user** and **create** and **register** the "**CarAdmin**" model and use:

* **list\_display** - including the **fields**: **model**, **year**, **owner**, and **car\_details**.

### Methods inside the CarAdmin model

**car\_details(obj: object)** is a **static** **method** and **returns** information about the **owner** and the **registration number** of all cars as follows:

* **"Owner: {owner\_name}, Registration: {registration\_number}"**
* If the **car** has no **owner,** the function should **return** "**No owner**".
* If the **car** has no **registration,** the function should **return** "**No registration number**".

Also, **include** a user-friendly **short description** for that method named "**Car Details**".

### Hint

**car\_details(obj)**: This is a custom method that takes an instance of the **Car** model (**obj**) and **returns** a string based on whether the **car** has an **owner** and a **registration**. It constructs a string that includes the owner's **name** and **registration number** or "**No owner**" and "**No registration number**" if the respective **fields** are missing.

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

**"car\_details.short\_description"**: This sets a custom column header for the "**car\_details()"** method in the **admin** list view. Once you're **logged** **in**, navigate to the list view of the "**Car"** model. You should see a custom column labeled "**Car Details**" that displays the information about the **owner** and **registration number** for each **car**, as specified in the "**car\_details()"** method.

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично