# Lab: Migrations and Django Admin

Submit your solutions to the SoftUni [**Judge system**](https://alpha.judge.softuni.org/Contests/Migrations-and-Django-Admin-Lab/4301).

For this lab, you are given an **ORM project skeleton** (you can download it from the current lesson's resources) with **one model** called **"Product"**. That model consists of the fields: **"name"**, **"description"**, **"price"**, **"category"**, and **"supplier"**.

## Migrations

**Note: The problem cannot be submitted in the Judge system.**

Your task is to help us **improve the model** by making some **changes to the code**:

* First, we want you to **migrate the model** to the database.
* Next, we need to **add a couple of records** to the created database table. To help us, you need to **open the** **caller.py** file and **execute the code on line 80**.
* We just remembered that we need to **put two more columns** in the table:
  + **created\_on** - a **date** **and** **time** **field**, that should be **set to the current date and time when the object is first created** and should **not** **be editable**.
  + **last\_edited\_on** - a **date** **and** **time** **field**, that should be **set to the current date and time every time the object is saved** and should **not** **be editable**.

**Add the fields to the model** and **migrate the changes**.

* We are willing to **add one more record**. **Run only the code on line 82** in the **caller.py** file.
* Now, we come up with the idea to **add the following field**:
  + **count** - a **positive integer field** with a **default value of 0** (zero).

We need you to **refactor the code** and **migrate the changes** to the database.

* We want to **add a couple of records**. **Run only the code on line 84** in the **caller.py** file.
* We saw that it was a poor idea and now we want you to **undo the changes from the previous task** (reverse the **last migration** and remove the **count** field).
* Now, we have come to the opinion that making the fields **"category"** and **"supplier"** **required** (rather than optional) is beneficial. **Make the** **changes to the code** and **migrate them** to the database.
* For now, we think we are satisfied with the model.

### Hint

**Note: Before start solving the problem,** **set up the database**.

First, we need to migrate the model to the database. We will use the terminal commands **"python manage.py makemigrations"** and **"python manage.py migrate"** successively.

Next, we will go to the **caller.py** file, and execute the code on **line 80**. If the products are added successfully to the database as record, we should **see the following output** on the console: A screenshot of a computer

Description automatically generated

We can also **check the table data**, using the PyCharm Database tool: A screenshot of a computer

Description automatically generated

Now, our next task is to add **two more fields** to the model and **migrate the changes**. We can see that, when we try to make the migration files, Django raises an error. That is because the database needs something to populate existing rows. We have **two options** - we can **provide a one-off value** that will be set only on the already existing rows, or we can **exit the operation and set a default valu**e in the model fields:A screenshot of a computer

Description automatically generated

Let us **choose option 1** for this example. We can **enter a default value** manually or **we can choose to use the now()** **function** provided by Django. Let us just **set the default value to now()** by pressing Enter:

A screenshot of a computer

Description automatically generated

If we **look at the migration file**, we ca see the default value added:  
A screenshot of a computer program

Description automatically generated

We can now easily migrate the changes to the database using the **"python manage.py migrate"** command.

Let us **add one more record** to the database using the code in the **caller.py**:  
A screenshot of a computer program

Description automatically generated

It is time to **execute our next task**. We need to **add a new field called count**: A screenshot of a computer program

Description automatically generated

**Migrate the changes** to the database. Then, execute only the **84th line**.

Unfortunatelly, we are given more work to do. Now, we need to **undo our changes**. First, let us **reverse the last migration** by using the command **"python manage.py migrate main\_app 0002"**:  
A screenshot of a computer

Description automatically generated

Next, we should **delete the field "count"**, so that it will not be added to the table as a column anymore. And, finally, we **do not need the migration file 0003, so we can delete it**. If we do not delete it, the next time we migrate the changes, Django will execute the migration file again.

Finally, **make the last changes** to the model and **migrate them**.

## Barcode System

**Note: The problem cannot be submitted in the Judge system.**

We've reconsidered it, and now we want to **make some more changes to the "Product" model**. Add a new **unique** **integer field** called **"barcode"** to the model. It will consist of the barcode of each product. For **all already applied products in the database, we should add the barcode value** - a **random unique number** from **100 000 000** to **999 999 999 both inclusive**.

### Hint

First, let us add the field **"barcode"** to the model:

A screenshot of a computer code

Description automatically generated

Next, let us **migrate the changes** to the database, and then on a separate migration file we can write our custom logic. First, we need to **open the terminal** and write the command **"python manage.py makemigrations --empty main\_app"** to generate a new empty migration file. Then, we can **populate the file**. Note, that we can use **random.sample(range, count)** Python function to get only random unique values:

## Register the Model

**Register the model** **"Product"** in the Django Admin site. **Open the interface** and **create one record** for each model. Familiarize yourself with the functionalities of the admin site. **Submit your project to the Judge system**.

### Hint

First, let us open the **admin.py** file in the **main\_app** and import the model "**Product"**. Now, we have two ways of registering the model in the admin site. One way to register the model is by directly providing the model. We use this when we **do not want to define any custom values** and **we are ok with the default admin interface**:A screen shot of a computer

Description automatically generated

Second way - we can **create a class** that **inherits from the ModelAdmin class** to register our model: A screenshot of a computer

Description automatically generated

To work with the admin site, we need to create a superuser. We achieve it by executing the terminal command **"python manage.py createsuperuser"**. Then, we need to pass the name and the password of the user.

## Customize the Admin

**Customize the interface** of the registered model. In a **ProductAdmin** class add the following functionality:

* **Display** the fields **"name"**, **"category"**, **"price"**, and **"created\_on"** (in that order) in the admin panel.
* Enable **searching** for **"name"**, **"category"**, and **"supplier"** fields in the admin panel.
* Create **filters** for **"category"** and **"supplier"** fields in the admin panel.
* **Control the layout** of **"Add"** and **"Change"** pages by grouping related fields within different sections:
  + Group **"General Information"** with fields **"name"**, **"description"**, **"price"**, **"barcode"**.
  + Group **"Categorization"** with fields **"category"** and **"supplier"**.
* Enable date-based drill-down navigation by the **"created\_on"** field.

### Hint

To include a date-based drill-down navigation by the field **created\_on**, we should use the option **date\_hierarchy** and set it to **"created\_on"** value:

A screenshot of a computer

Description automatically generated