Lab: Migrations and Django Admin

This document defines the problems for the in-class lab for the **Python ORM course** @ SoftUni Global.

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"`**.

1. 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).

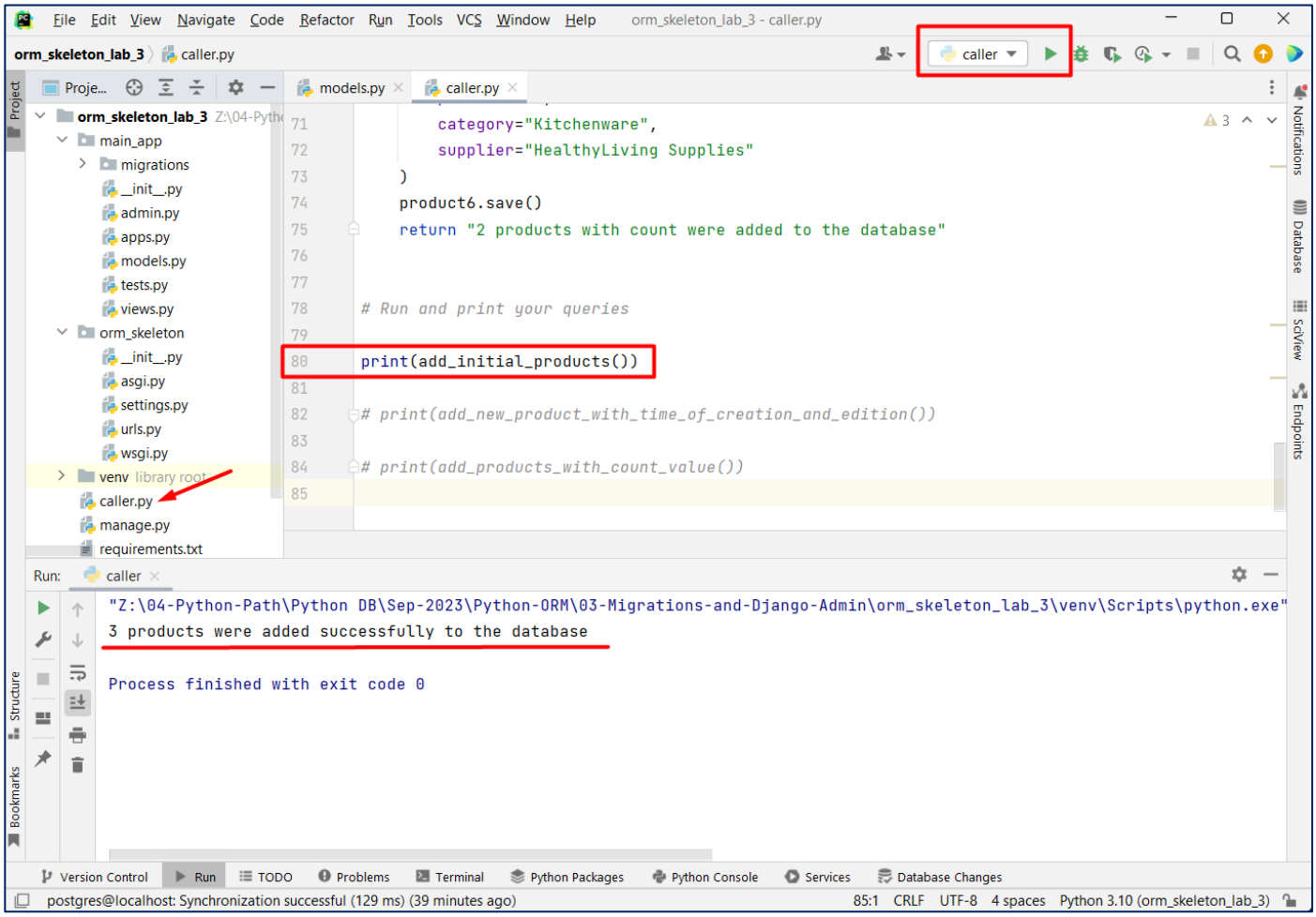
  We need you to **refactor the code** and **migrate the changes** to the database.
- We want to **add a couple of records**. **Run only the code on line 84** in the **`caller.py`** file.
- We saw that it was a poor idea and now we want you to **undo the changes from the previous task** (reverse the **last 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

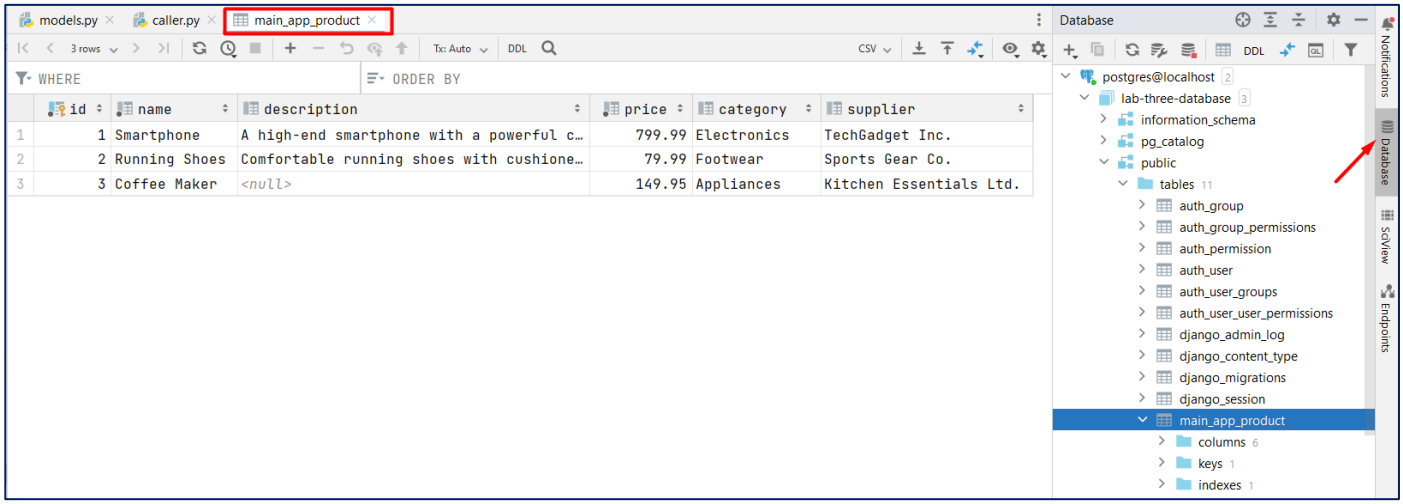
Note: Before start solving the problem, set up the database.

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:

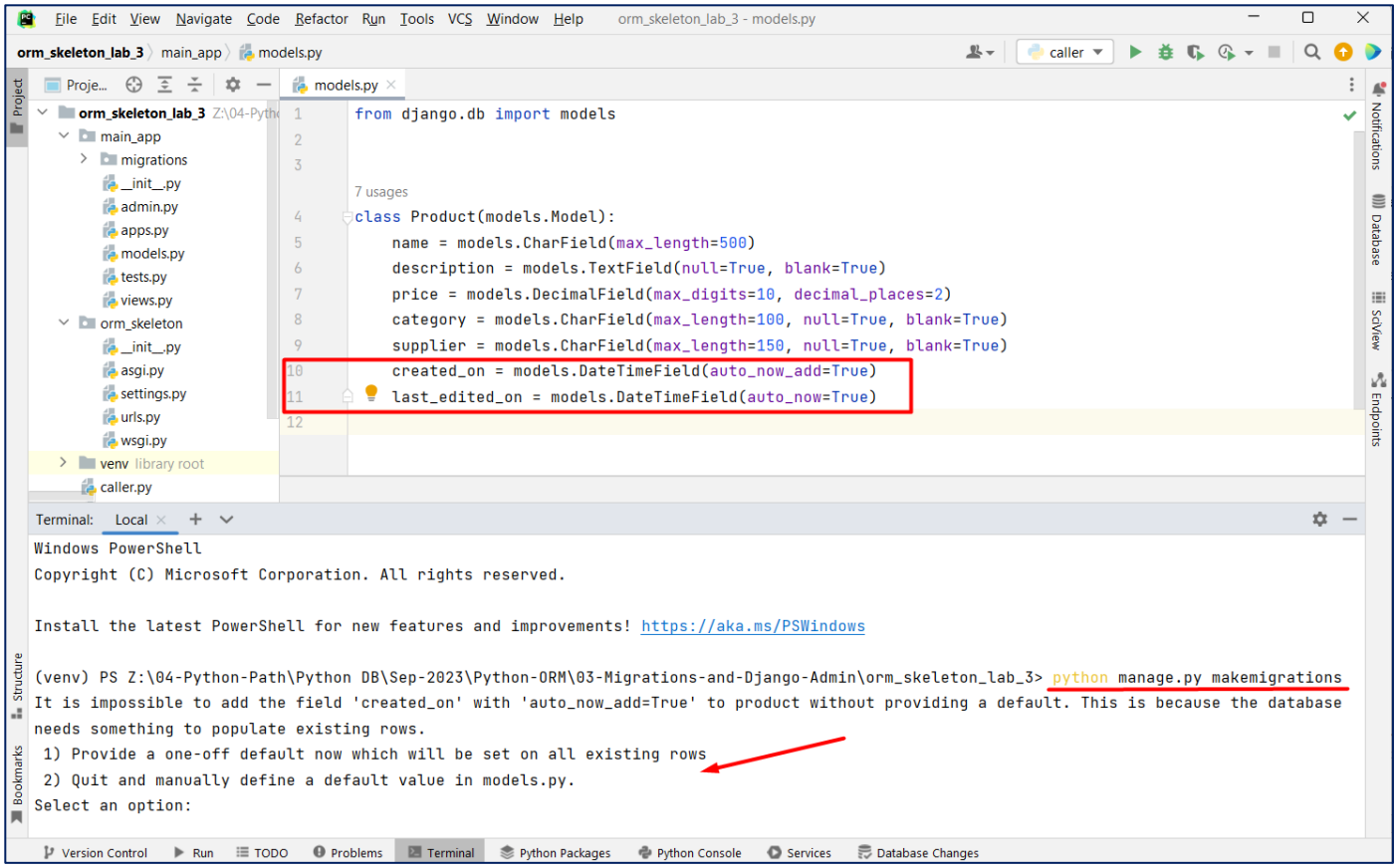


We can also **check the table data**, using the PyCharm Database tool:

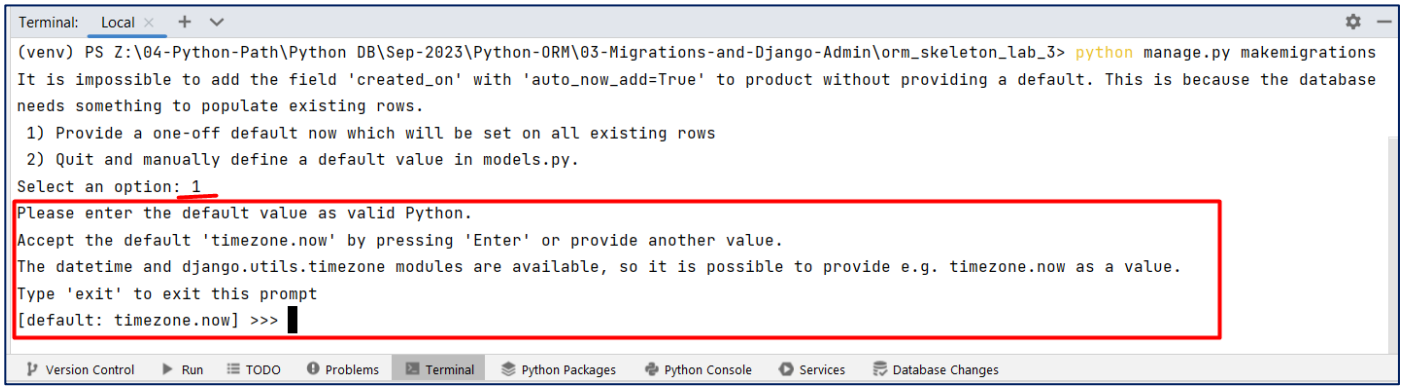


Now, our next task is to add **two more fields** to the model and **migrate the changes**. We can see that, when we try to make the migration files, Django raises an error. That is because the database needs something to populate existing rows. We have **two options** - we can **provide a one-off value**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

rows, or we can **exit the operation and set a default value** in the model fields:



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:



If we **look at the migration file**, we ca see the default value added:

```
models.py  0002_product_created_on_product_last_edited_on.py

class Migration(migrations.Migration):

    dependencies = [
        ('main_app', '0001_initial'),
    ]

    operations = [
        migrations.AddField(
            model_name='product',
            name='created_on',
            field=models.DateTimeField(auto_now_add=True, default=django.utils.timezone.now),
            preserve_default=False,
        ),
        migrations.AddField(
```

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**:

```
models.py  caller.py

    return "2 products with count were added to the database"


# Run and print your queries

# print(add_initial_products())

print(add_new_product_with_time_of_creation_and_edition())

# print(add_products_with_count_value())

Run:  caller

"Z:\04-Python-Path\Python DB\Sep-2023\Python-ORM\03-Migrations-and-Django-Admin\orm_sk
1 product with time of creation and edition was added to the database

Process finished with exit code 0
```

It is time to **execute our next task**. We need to **add a new field called count**:

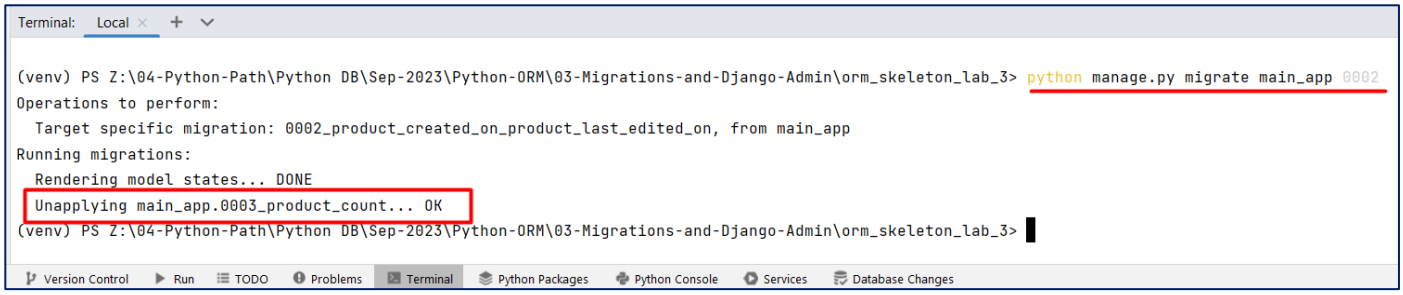
```python
from django.db import models


7 usages
class Product(models.Model):
    name = models.CharField(max_length=500)
    description = models.TextField(null=True, blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    category = models.CharField(max_length=100, null=True, blank=True)
    supplier = models.CharField(max_length=150, null=True, blank=True)
    created_on = models.DateTimeField(auto_now_add=True)
    last_edited_on = models.DateTimeField(auto_now=True)
    count = models.PositiveIntegerField(default=0)

```

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"**:



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**.

2. 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:

models.py

```python
from django.db import models


7 usages
class Product(models.Model):
    name = models.CharField(max_length=500)
    description = models.TextField(null=True, blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    category = models.CharField(max_length=100)
    supplier = models.CharField(max_length=150)
    created_on = models.DateTimeField(auto_now_add=True)
    last_edited_on = models.DateTimeField(auto_now=True)
    barcode = models.IntegerField()

```

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:

0005_auto_20230910_1737.py

```python
import random
from django.db import migrations


1 usage
def add_barcode(apps, schema_editor):
    Product = apps.get_model("main_app", "Product")
    all_products = Product.objects.all()
    all_barcodes = random.sample(
        range(100000000, 1000000000),
        len(all_products))
    for i in range(len(all_products)):
        product = all_products[i]
        product.barcode = all_barcodes[i]
        product.save()


1 usage
def reverse_add_barcode(apps, schema_editor):
    Product = apps.get_model("main_app", "Product")
    for product in Product.objects.all():
        product.barcode = 0
        product.save()


class Migration(migrations.Migration):
    dependencies = [('main_app', '0004_product_barcode')]
    operations = [migrations.RunPython(add_barcode, reverse_code=reverse_add_barcode)]

```

3. 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

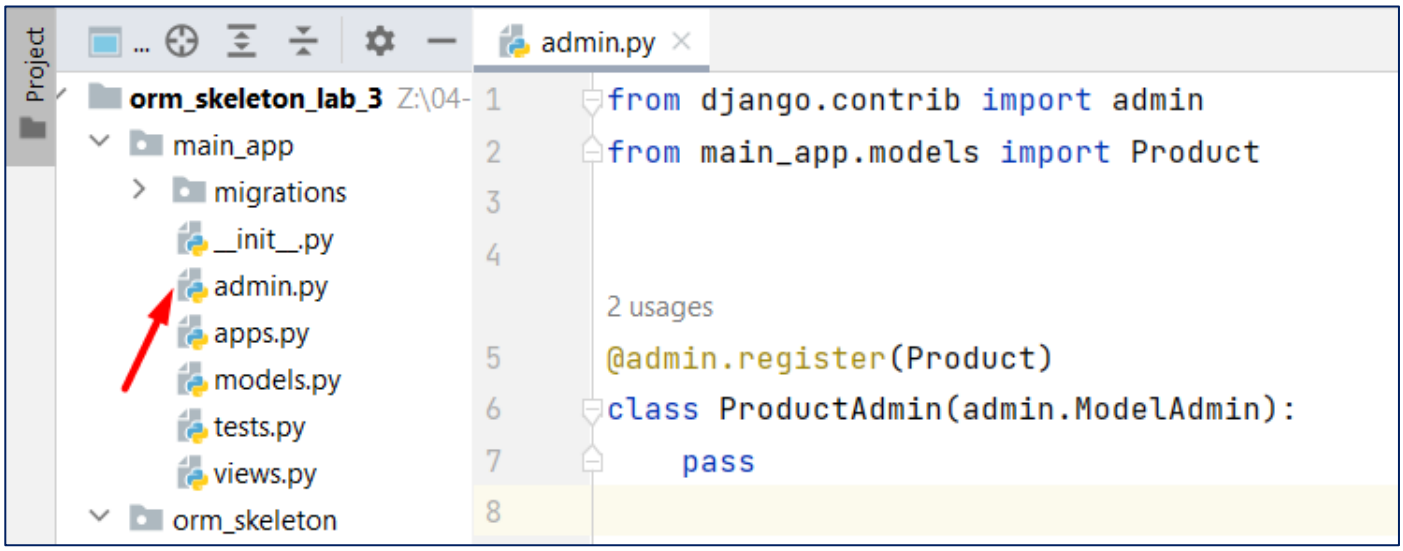
First, let us open the **admin.py** file in the **main_app** and import the model "**Product**". Now, we have two ways of registering the model in the admin site. One way to register the model is by directly providing the model. We use this when we **do not want to define any custom values** and **we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

```python
from django.contrib import admin
from main_app.models import Product


admin.site.register(Product)

```

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



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.

4. 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:

admin.py

```python
from django.contrib import admin
from main_app.models import Product


2 usages
@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = ('name', 'category', 'price', 'created_on')
    search_fields = ('name', 'category', 'supplier')
    list_filter = ('category', 'supplier')
    fieldsets = (
        ('General Information', {
            'fields': ('name', 'description', 'price', 'barcode')
        }),
        ('Categorization', {
            'fields': ('category', 'supplier')
        }),
    )
    date_hierarchy = 'created_on'

```