# Lab: Advanced Queries in Django

This document defines the problems for the in-class lab for the [Python ORM course @ Software University](https://softuni.bg/trainings/4253/python-orm-october-2023).

Submit your solutions in the SoftUni [Judge system](https://judge.softuni.org/Contests/4333/Advanced-Queries-in-Django-Lab).

For this lab, you are given an **ORM project skeleton** (you can download it from [here](https://softuni.bg/downloads/svn/python-db/trunk/Sep-2023/Python-ORM/09-Advanced-Queries-in-Django/09-lab-orm-skeleton.zip)) containing **a Shop Management System** with **products, categories, customers**,and **orders**.

In the **caller.py** file, you will find a function called **"add\_records\_to\_database"** on line 62. **Execute the function** to populate the tables with the needed data. Remember to **return it as a comment**, so you do not overpopulate the tables while solving the problems**.**

## Available Products

Create a custom manager called **"ProductManager"** that extends the built-in model manager with two methods:

* **available\_products** - returns all **products** that are **currently available**.
* **available\_products\_in\_category(category\_name: str)** - returns all **products in a category** that are **currently available**.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| print('All Products:')  print(Product.objects.all())  print()  print('All Available Products:')  print(Product.objects.available\_products())  print()  print('All Available Food Products:')  print(Product.objects.available\_products\_in\_category("Food")) |
| **Output** |
| All Products:  <QuerySet [<Product: Food: Pizza>, <Product: Food: Burger>, <Product: Food: Apples>, <Product: Food: Bread>, <Product: Food: Pasta and Sauce Bundle>, <Product: Food: Tomatoes>, <Product: Food: Carton of Eggs>, <Product: Food: Cheddar Cheese>, <Product: Food: Milk>, <Product: Drinks: Coca Cola>, <Product: Drinks: Orange Juice>, <Product: Drinks: Bottled Water>, <Product: Drinks: Orange Soda>, <Product: Drinks: Bottled Green Tea>, <Product: Drinks: Beer>]>  All Available Products:  <QuerySet [<Product: Food: Apples>, <Product: Food: Bread>, <Product: Food: Tomatoes>, <Product: Food: Carton of Eggs>, <Product: Food: Milk>, <Product: Drinks: Coca Cola>, <Product: Drinks: Bottled Water>, <Product: Drinks: Orange Soda>, <Product: Drinks: Beer>]>  All Available Food Products:  <QuerySet [<Product: Food: Apples>, <Product: Food: Bread>, <Product: Food: Tomatoes>, <Product: Food: Carton of Eggs>, <Product: Food: Milk>]> |

## Product Quantity Ordered

Create a function called **"product\_quantity\_ordered"** that returns a summary of the **total quantity ordered** for **each** **product** available in the store in the given format below:  
**"Quantity ordered of {product\_name1}: {total\_ordered\_quantity1}**

**…**

**Quantity ordered of {product\_nameN}: {total\_ordered\_quantityN}"**Return only the information for **products that have at least one unit ordered**. Arrange the information in **descending** **order** **based on the total quantity ordered**.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| print(product\_quantity\_ordered()) |
| **Output** |
| Quantity ordered of Carton of Eggs: 8  Quantity ordered of Apples: 4  Quantity ordered of Bottled Water: 4  Quantity ordered of Tomatoes: 1  Quantity or ordered edered of Milk: 1  Quantity ordered of Orange Soda: 1 |

## Ordered Products Per Customer

Create a function called **"ordered\_products\_per\_customer"** that returns a summary of **each ordered** **product** **by each customer** in the given format below:  
**"Order ID: {order\_id1}, Customer: {customer\_username1}**

**- Product: {product\_name1}, Category: {category\_name1}**

**…**

**- Product: {product\_name1}, Category: {category\_name1}**

**…**

**Order ID: {order\_idN}, Customer: {customer\_usernameN}"**

Arrange the information in **ascending** **order** **by the order ID**.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| print(ordered\_products\_per\_customer()) |
| **Output** |
| Order ID: 1, Customer: john\_doe  - Product: Apples, Category: Food  - Product: Tomatoes, Category: Food  - Product: Carton of Eggs, Category: Food  - Product: Orange Soda, Category: Drinks  Order ID: 2, Customer: peter132  - Product: Apples, Category: Food  - Product: Milk, Category: Food  Order ID: 3, Customer: john\_doe  - Product: Bottled Water, Category: Drinks  - Product: Carton of Eggs, Category: Food |

## Available Products Prices

Create a function called **"filter\_products"** that returns information for **all available products with prices greater than 3.00** in the store in the format:  
**"{product\_name1}: {product\_price1}lv.**

**…**

**{product\_nameN}: {product\_priceN}lv.}"**

Arrange the information in **descending** **order** **by the price**. If there are **two or more products with the same price**, order them by **name in ascending order** (alphabetically).

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| print(filter\_products()) |
| **Output** |
| Beer: 5.49lv.  Orange Soda: 5.49lv.  Bottled Water: 4.99lv.  Apples: 3.99lv.  Carton of Eggs: 3.49lv.  Milk: 3.49lv. |

## Give Discounts

Create a function called **"give\_discount"** that reduces the product's price by 30% of **all available products with prices greater than 3.00**. Then, it returns information about **all available products** and **their prices** in the following format:  
**"{product\_name1}: {product\_price1}lv.**

**…**

**{product\_nameN}: {product\_priceN}lv.}"**

Arrange the information in **descending** **order** **by the price**. If there are **two or more products with the same price**, order them by **name in ascending order** (alphabetically).

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| print(give\_discount()) |
| **Output** |
| Beer: 3.84lv.  Orange Soda: 3.84lv.  Bottled Water: 3.49lv.  Tomatoes: 2.99lv.  Apples: 2.79lv.  Bread: 2.49lv.  Carton of Eggs: 2.44lv.  Milk: 2.44lv.  Coca Cola: 1.99lv. |