**Python ORM - Exam Prep II**

Submit your solutions to the SoftUni [**Judge System**](https://judge.softuni.org/Contests/4336/Exam-Prep-II).

*Create three models that allow you to manage, manipulate, and extract data from the Database. These models together form the basis of a simple e-commerce system.*

*Your project will manage user profiles, orders, and products.*

# **Skeleton**

You are provided with a ready-to-use skeleton. Do not change the folder and file names! You are allowed to add additional files.

**Judge Submissions:**

* Once you have **completed a task**, you must **archive** the project files (**zip** format) and upload the zip file to the contest (for the **corresponding judge task**). You **do not need to include** your **venv**, **.idea**, **pycache**, and **\_\_MACOSX** (for Mac users), so you do **not exceed** the maximum allowed size of **31.25** **KB**
* Submit a solution (archived project files) for **each** **task**!

A screenshot of a computer

Description automatically generated

# **Database – 100 points**

You will need to create **three models** in the **models.py** file:

### **Profile Model**

* + **full\_name**
    - A **character** field.
    - Represents the full name of the user.
    - Validation: **Minimum length** of **2** characters, **maximum length** of **100** characters.
* **email**
  + - An **email** field.
    - Represents the email address of the user.
* **phone\_number**
  + - A **character** field.
    - Represents the phone number of the user.
    - Validation: **Maximum length** of **15** characters.
    - Additional Note: This field is typically a string to accommodate various phone number formats.
* **address**
  + - A **text** field.
    - Represents the address of the user.
    - Additional Note: This field can store longer text, suitable for addresses.
* **is\_active**
  + - A **boolean** field.
    - Indicates whether the profile is active or not.
    - Default Value: **True** (Active).
* **creation\_date**
  + - A **date/time** field.
    - **Automatically** records the **date** and **time** when the **profile** is **created.**

### **Product Model**

* **name**
* A **character** field.
* Represents the name of the product.
* Validation: **Maximum length** of **100** characters.
* **description**
  + A **text** field.
* Provides a detailed description of the product.
* **price**
  + A **decimal** field**.**
* Represents the price of the product.
* Validation: **Maximum** **digits**: **10**, **decimal** **places**: **2**. **Minimum value** of **0.01** to ensure it's a positive value.
* **in\_stock**
  + A **positive** **integer** field.
* Represents the quantity of the product in stock.
* Validation: **Minimum value** of **0**
* **is\_available**
  + A **boolean** field.
* Indicates whether the product is currently available for purchase.
* Default Value: **True** (Available).
* **creation\_date**
  + A **date/time** field.
* **Automatically** records the **date** and **time** when the **product** is **created**.

### **Order Model**

* **profile**
  + A **foreign key** to the **Profile** model.
* Establishes a **many-to-one** relationship with the Profile model, associating each order with a user's profile.
* **ON DELETE** constraint must be set to **CASCADE**
* **products**
  + A **many-to-many** field to **Product** model.
* Establishes a **many-to-many** relationship with the Product model, allowing an order to contain multiple products and a product to be in multiple orders.
* **total\_price**
  + A **decimal** field.
* Represents the total price of the order.
* Validation: **Maximum digits**: **10**, **decimal places**: **2**. **Minimum value** of **0.01** to ensure it's a positive value.
* **creation\_date**
  + A **date/time** field.
* **Automatically** records the **date** and **time** when the **order** is **created**.
* **is\_completed**
* A **boolean** field.
* Indicates whether the order has been completed or not.
* Default Value: **False** (Not Completed).

**Hint**: You can use model Mixins to stick to good practices and avoid code repeating (DRY principle).

# **Customizing Django Admin Site – 30 points**

Register your models to the Django Admin Site (**admin.py** file) and make the following customizations which will enhance the admin interface by providing more meaningful and searchable information:

## **ProfileAdmin**

* **Display fields**: Specify the fields to be displayed in the list view of the admin site for the **Profile model**.
  + Fields: **'full\_name'**, **'email'**, **'phone\_number'**, **'is\_active'**
* **Search fields**: Enables search by **full\_name** and **email** in the admin site.

## **ProductAdmin**

* **Display fields**: Specify the fields to be displayed in the list view of the admin site for the **Product model**.
  + Fields: **'name'**, **'price'**, **'in\_stock'**, **'is\_available'**
* **Filters**: Add a filter for **is\_available** in the admin site.
* **Search fields**: Enable search by **name** in the admin site.

## **OrderAdmin**

* **Display fields**: Specify the fields to be displayed in the list view of the admin site for the **Order model**.
  + Fields: **'profile', 'total\_price', 'creation\_date', 'is\_completed'**
* **Filters**: Add a filter for **is\_completed** in the admin site.
* **Search fields**: Enable search by **profile’s full\_name** (searching orders by profile's full name).

# **Custom Model Manager – 20 points**

Create a **custom model manager** for the **Profile** **model** and add your **custom method**:

## **get\_regular\_customers()**

This method **retrieves** and **returns** all **profile objects** with **more than** **two orders. Order** profiles **by number of orders**, **descending**. You should count **all** orders **regardless of** their **status ("Completed"** or **"Not Completed")**.

# **Django Queries I – 75 points**

In the **caller.py** file create the following functions:

## **get\_profiles(search\_string=None)**

This function accepts the following argument with default **None** value:

* **search\_string** – string value or **None**

It **retrieves** profile objects by **partially** and **case-insensitively** matching the given searching criteria for **full name**, **email**, or **phone number**. Check if **any** of these three field values (**full name**, **email**, or **phone number**) contain the search string.

**Order** the profile objects by **full name**, **ascending.**

**Return** a **string** in the following format, each profile info on a new line:

**"Profile: {full\_name1}, email: {email1}, phone number: {phone\_number1}, orders: {num\_of\_orders1}**

**Profile: {full\_name2}, email: {email2}, phone number: {phone\_number2}, orders: {num\_of\_orders2}**

**…**

**Profile: {full\_nameN}, email: {emailN}, phone number: {phone\_numberN}, orders: {num\_of\_ordersN}"**

* If no profiles match the criteria, **return** an **empty string ("")**.
* **Hint:** You can use Q objects but first check if the search string is not None.

## **get\_loyal\_profiles()**

This function accepts no arguments.

It **retrieves** profile objects with **more than** **two orders, ordered by number of orders, descending.** You should count **all** orders **regardless of** their **status ("Completed"** or **"Not Completed")**.

**Return** a **string** in the following format, each profile info on a new line:

**"Profile: {full\_name1}, orders: {num\_of\_orders1}**

**Profile: {full\_name2}, orders: {num\_of\_orders2}**

**…**

**Profile: {full\_nameN}, orders: {num\_of\_ordersN}"**

* If no profiles match the criteria, **return** an **empty string ("")**.
* **Hint:** You can use the custom model manager method.

## **get\_last\_sold\_products()**

This function accepts no arguments.

It **retrieves** the **products** from the **latest order object**, **ordered by product name**, **ascending.** The status of the order does not matter **("Completed"** or **"Not Completed")**.

**Return** a **string** in the following format:

**"Last sold products: {product\_name1}, {product\_name2}, … {product\_nameN}"**

* **Product names** must be **separated** by a **comma and space (", ")**
* If there are **no orders** and respectively **no products sold**, **return** an **empty string ("")**.

# **Django Queries II – 75 points**

## **get\_top\_products()**

This function accepts no arguments.

It **retrieves** the **most frequently sold** **products** from all orders. **Order** them **by the number of times** the **product** has been **sold** (included in an order), **descending**, **then** **ascending** by **product** **name**. The status of the orders does not matter **("Completed"** or **"Not Completed")**.

Take the **top five ordered products**.

**Return** a **string** in the following format:

**"Top products:**

**{product\_name1}, sold {num\_orders1} times**

**{product\_name2}, sold {num\_orders2} times**

**…**

**{product\_name5}, sold {num\_orders5} times"**

* **Product name** and **sales info** must be **separated** by a **comma and space (", ")**. **Each** **product** info is on a **new line**.
* In case the **sold items** are **less than five** in total, **return all of them**, **ordered** as described.
* If there are **no orders** and respectively **no products sold**, **return** an **empty string ("")**.

## **apply\_discounts()**

This function accepts no arguments.

It **retrieves** **order** **objects** that have **more than two products**, whose **status** is "**Not Completed**" (**is\_completed=False**), and applies a **discount** of **10%** to the **total price**.

**Return** a **string** in the following format:

**"Discount applied to {num\_of\_updated\_orders} orders."**

* If **no orders** are **affected**, the value of "**num\_of\_updated\_orders**"will be **0 (zero)**.
* **Hint**: You can use F object to efficiently update the total price for all selected orders.

## **complete\_order()**

This function accepts no arguments.

It **retrieves** the **first** (oldest) **order** **object** from your database whose **status** is "**Not Completed**" and **changes** it from "**Not Completed**" (**is\_completed=False**) to "**Completed**" (**is\_completed=True**).

Remember that you must **decrease** the **quantity** of the **ordered products** you have in stock (**in\_stock**).

If a quantity becomes **0** (**zero**), **change** the **status** of the **product** to "**Not Available**" (**is\_available=False**).

**Return** a **string** in the following format:

**"Order has been completed!"**

* If there are **no orders** or **all** **orders** have been **completed**, **return** an **empty string ("")**.

# **Testing Data Constraints**

* There will always be active profiles, enough quantity, and available products when making orders.
* The following outputs show the **expected behavior** of the functions. Populate the database with your own testing data and then check if the functions produce the expected results.

# **Examples**

|  |
| --- |
| **Test Code** |
| print(Profile.objects.get\_regular\_customers()) |
| **Output** |
| <QuerySet [<Profile: Adam Smith>]> |
| **Test Code** |
| print(get\_profiles('Co')) |
| **Output** |
| Profile: Adam Smith, email: as@test.com, phone number: 001 555 555, orders: 3  Profile: Susan James, email: sj@test.co.uk, phone number: 0044 333 222, orders: 1 |
| **Test Code** |
| print(get\_profiles('9zz')) |
| **Output** |
|  |
| **Test Code** |
| print(get\_loyal\_profiles()) |
| **Output** |
| Profile: Adam Smith, orders: 3 |
| **Test Code** |
| print(get\_last\_sold\_products()) |
| **Output** |
| Last sold products: Desk M, Display DL |
| **Test Code** |
| print(get\_top\_products()) |
| **Output** |
| Top products:  Display DL, sold 3 times  Desk M, sold 2 times  Printer Br PM, sold 2 times |
| **Test Code** |
| print(apply\_discounts()) |
| **Output** |
| Discount applied to 1 orders. |
| **Test Code** |
| print(complete\_order()) |
| **Output** |
| Order has been completed! |