Remarcable Project SQL Table Design

3 Tables/Django Models:

1. Product
2. Category
3. Tag

Constraints

1. You should be able to assign a single category to any given product
2. You can assign multiple tags to multiple products
3. You should be able to assign the same tags to other products

Example

- **Product**: iPhone 13
- **Category**: Phone
- **Tags:** Smart Phone, 128GB, Camera, 6.1”

Database Table Set-Up

- 4 tables:
    - Product
    - Category
    - Tags
    - Tag/Product Relationship
    - Search History

- There are a couple ways to assign tags to each product:
    1. Add an extra table with tag/product relationships (proper use of foreign keys)
        a. This table does not scale well but keeps the proper relationships and is relatively straightforward to query.
    2. Add tags to another column as an array (containing tag_id) or as a string with comma separated tags
        a. This option may scale well but requires additional functions with each query to parse the array or string and return the proper tags.

- For ease of querying and implementation, I decided to go with option 1
- The 5th database table is to record the user’s search history. This table just stores the string string of the previous searches so that in the case that the search page is refreshed or we want to filter the search results, we can reload the correct table resulting from the user’s search.

Product

product_id [pk]	product_name	category [fk]
1	iPhone13	1
2	Samsung Galaxy 21	1
3	WD Hard Drive	2

Category

category_ id [pk]	category_name
1	Phone
2	Hard Drive

Tag

tag_id [pk]	tag_name
1	Smart Phone
2	128GB
3	Camera
4	6.1”

Tag/Product Relationship

product_id [fk]	Tag_id [fk]
1	1
1	2
1	3
1	4
2	1

2	3
3	2

Search History

id [pk]	**search_name**
1	Iphone hard drive
2	Graphics card processors
3	Hard drive gpu

Page Set-Up

- Since this is a simple application just to demonstrate retrieving data from a database, we utilize two 3 html files
    - **Base.html**
        - This defines the basic structure of every web page such as the search bar and home page link
    - **Home.html**
        - This page shows all of the filters, pulling from the categories and tag tables in the database. It also shows every product entry in the database.
    - **Search.html**
        - This page is only shown if the user utilizes the search bar. The search page will return matching results and allow the user to filter these results. If nothing is returned, that means that the search returned zero matching results.

SQL Queries

- Using Django, it is very inefficient to write raw SQL queries everytime you want to access a table in the database. In this project, there are many redundant queries that also need further data manipulation. I decided it was best to define most of these django query functions within its own file: **query_functions.py**
- There are a few additional query functions within the **views.py** file.
- These query functions are listed in the order in which they appear in each file. Additional comments can be seen within the python file itself.

query_functions.py

Django Query Function: product_table = Product.objects.select_related('category')
Raw SQL:
SELECT "remarcable_app_product"."id", "remarcable_app_product"."product_name", "remarcable_app_product"."category_id", "remarcable_app_category"."id", "remarcable_app_category"."category_name" FROM "remarcable_app_product" INNER JOIN "remarcable_app_category" ON ("remarcable_app_product"."category_id" = "remarcable_app_category"."id")

- Select_related performs an INNER JOIN where there a foreign key relationships between tables.

Django Query Function:
tag_table = TagProductRelationship.objects.select_related('product','tag')

Raw SQL:
SELECT "remarcable_app_tagproductrelationship"."id", "remarcable_app_tagproductrelationship"."product_id", "remarcable_app_tagproductrelationship"."tag_id", "remarcable_app_product"."id", "remarcable_app_product"."product_name", "remarcable_app_product"."category_id", "remarcable_app_tag"."id", "remarcable_app_tag"."tag_name" FROM "remarcable_app_tagproductrelationship" INNER JOIN "remarcable_app_product" ON ("remarcable_app_tagproductrelationship"."product_id" = "remarcable_app_product"."id") INNER JOIN "remarcable_app_tag" ON ("remarcable_app_tagproductrelationship"."tag_id" = "remarcable_app_tag"."id")

Django Query Function:
category_data = Category.objects.values_list('category_name', flat=True)

Raw SQL:
SELECT "remarcable_app_category"."category_name" FROM "remarcable_app_category"

-values_list() pulls the raw values from a table and stores them in a python list for easier accessibility.

Django Query Function:
```
tag_data = Tag.objects.values_list('tag_name', flat=True)
```

Raw SQL:

SELECT "remarcable_app_tag"."tag_name" FROM "remarcable_app_tag"

Django Query Function:

```
tag_name = Tag.objects.get(tag_name = tag_filter)
```

Raw SQL:
SELECT tag_name FROM remacable_app_tag WHERE tag_name = tag_filter

- get() returns the matching object from the table where the lookup condition is True

Django Query Function:

```
category_name = Category.objects.get(category_name = category_filter)
```

Raw SQL:
SELECT category_name FROM remarcable_app_category WHERE category_name = category_filter

Django Query Function:

```
product_table = product_table.filter(category_id = category_name.id)
```

Raw SQL:
SELECT * FROM product_table WHERE category_id = category_name.id

- filter() returns a new table/QuerySet where containing only objects where the given condition is True

Django Query Function: (there are many duplicates/variations of this function)

```
temp_product = product_table.filter(product_name__icontains=term).values_list('id', flat=True)
```

Raw SQL:
SELECT * FROM product_table WHERE product_name ILIKE %term%

- Icontains is non-casesensitive. The “%” around term means that any number of characters in a row can satisfy the condition.

Django Query Function:

```
SearchHistory.objects.first().delete()
```

Raw SQL:
DELETE FROM remaracble_app_searchhistory WHERE id = 1

views.py

Django Query Function:
latest_search = SearchHistory.objects.create(search_name=raw_search)

Raw SQL:
INSERT INTO ramarcable_app_searchhistory (id, search_name) VALUES (1,raw_search)

Django Query Function:
SearchHistory.objects.all().values_list()

Raw SQL:
SELECT * FROM remarcable_app_searchhistory