Working with Queries in Django

SoftUni Team

Technical Trainers

Software University

https://softuni.bg

Have a Question?

sli.do

#python-db

Table of Contents

Useful Methods

Filtering, Excluding, Ordering Data

Methods in Django Queries

- We use **methods** for
  - **filtering**, **excluding**, **ordering**, and **counting** data in the database
- They provide **powerful functionality** for
  - **querying** and **manipulating** data
  - using Django's **ORM** (Object-Relational Mapping) **capabilities**

Filtering Data

- `filter()` method
  - Retrieves a **subset** of **objects** from a database
  - Takes **one** or **more** **keyword arguments**
  - Each argument represents a **field** and its **corresponding value** to **filter** against
  - Returns a **QuerySet**
    - Containing **objects** that **match** the **specified conditions**

Filtering Data

- Using **filter()** method

caller.py

```
def filter_employees():
   filtered_employees = Employee.objects.filter(job_level='Sr.')
   ...
```

Keyword argument

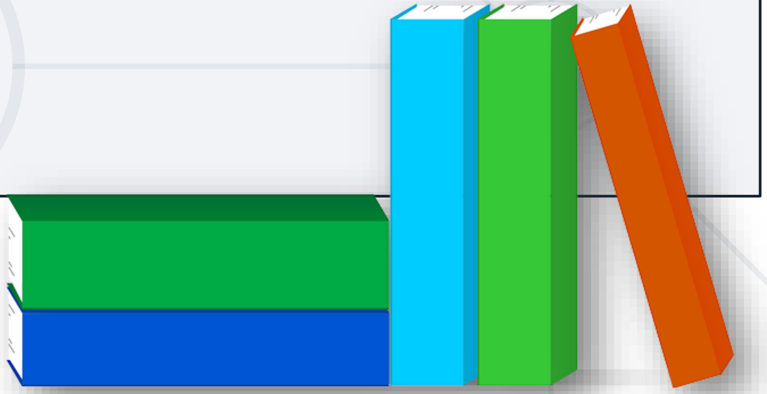
Filter all employee objects that match the condition

Problem: Books Finder

- You are given an **ORM project skeleton** (you can download it from here) with **three models**: **"Author"**, **"Book"**, and **"Review"**
- Create a function called **"find_books_by_genre_and_language"** that:
  - **Receives a book genre and a book language** as arguments
  - **Returns a queryset** of all books that concurrently satisfy **both** specified criteria - **genre and language**

Solution: Books Finder

```python
# caller.py
def find_books_by_genre_and_language(book_genre, book_language):
    found_books = Book.objects.filter(
        genre=book_genre,
        language=book_language
    )
    return found_books
```



Excluding Data

- `exclude()` method
  - Retrieves a **subset** of **objects** from a database
  - Takes **one** or **more** **keyword arguments**
  - Each argument represents a **field** and its **corresponding value** to **exclude**
  - Returns a **QuerySet**
    - containing **objects** that **do not match** the **specified conditions**, **excluding** the matching ones

Excluding Data

- Using **exclude()** method

```
caller.py
```

```python
def exclude_employees():
    excluding_employees = Employee.objects.
exclude(job_level='Sr.')
    ...
```

Keyword argument

Exclude all employee objects that match the condition

Problem: Find Authors' Nationalities

- Create a function called "**`find_authors_nationalities`**" that:
  - Finds all **authors** whose **nationalities** are **NOT null**
  - **Returns information** about each of them in the format:

```
"{first_name} {last_name} is {nationality}"
...
"{first_name} {last_name} is {nationality}"
```

Solution: Find Authors' Nationalities

```python
#caller.py
def find_authors_nationalities():
    found_authors = Author.objects.exclude(nationality=None)

    result = [
        f"{a.first_name} {a.last_name} is {a.nationality}"
        for a in found_authors
    ]

    return "\n".join(result)
```

Ordering Data

- `order_by()` method
  - Retrieves objects from the database in a **specific order**
  - Takes **one** or **more** **field names** as **arguments**
  - Returns a **QuerySet**
    - **sorted** based on **field names**
    - a **hyphen** ("**-**") prefix sorts in **descending** **order**

Ordering Data

- Using **order_by()** method

caller.py

```
def order_employees():
    ordered_employees = Employee.objects.
order_by('last_name')
    ...
```

Field name

Retrieve all employee objects ordered by last_name in ascending order

Ordering Data in Descending Order

- Using **order_by()** method to order data **descending**

caller.py

```
def order_desc_employees():
    ordered_desc_employees =
Employee.objects.order_by('-last_name')
    ...
```

Field name

Retrieve all employee objects ordered by last_name in descending order

Problem: Order Books by Year

- Create a function called **"`order_books_by_year`"** that:
  - **Orders all books** by their **publication year** in **ascending order**. If there are **two or more books published** in the **same year**, order them by **title in ascending order** (alphabetically)
  - **Returns information** about each book in the format:

```
"{publication_year} year: {title} by {author}"
...
"{publication_year} year: {title} by {author}"
```

Solution: Order Books by Year

```python
# caller.py
def order_books_by_year():
    ordered_books = Book.objects.order_by("publication_year",
                                          "title")

    result = [
        f"{b.publication_year} year: {str(b)}"
        for b in ordered_books
    ]
    return "\n".join(result)
```

Book model __str__ method

Counting Records in Database

- `count()` method
  - Retrieves the **number** of objects that **match** a specific **query** or **filter condition**
  - Available on a **QuerySet**
  - Returns an **integer**
    - representing **the number** of objects that **match** the query
  - Does **not** retrieve the **actual objects** themselves

Counting Records in Database

- Using **count()** method

caller.py

```
def count_employees():
    number_of_employees = Employee.objects.count()
    ...
```

Retrieve the number of all employee objects

count() vs len()

- `count()`
  - **Only** retrieves the **count**, **not** the **actual objects**
  - Provides an **efficient way** to **calculate** the **count** of objects on the **database side**
- `len()`
  - Retrieves **all objects** and calculates the **count** on the **Python side**
  - **Less efficient** for **large** datasets

Selecting a Single Object

- `get()` method
  - Retrieves a **single object** that **matches** the specified query **criteria**
  - Accepts **one** or **more keyword arguments** as query **criteria**
  - Raises an **exception** if **no object** has been found
    - **`DoesNotExist`** Exception
  - Raises an **exception** if **multiple objects** have been found
    - **`MultipleObjectsReturned`** Exception

Selecting a Single Object

- Using **get()** method

caller.py

```
def get_employee_by_id():
    employee = Employee.objects.get(id=2)
    print(employee.pk)
    print(employee.id)
    ...
```

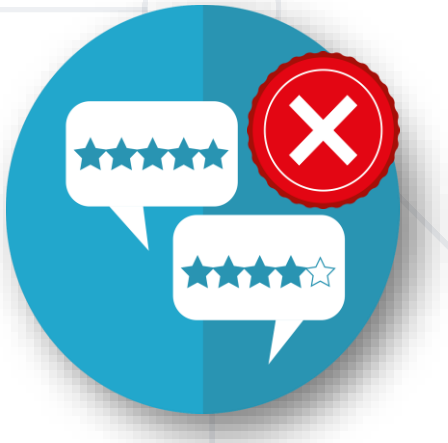
You will often see "pk" instead of "id"

Retrieve the employee with id=2

Problem: Delete Review by ID

- Create a function called "`delete_review_by_id`" that
  - **Receives a review's ID** as an argument
  - **Deletes the review's record** by the given ID
  - **Returns information** about the deleted review in the format:

```
"Review by {reviewer_name} was deleted"
```



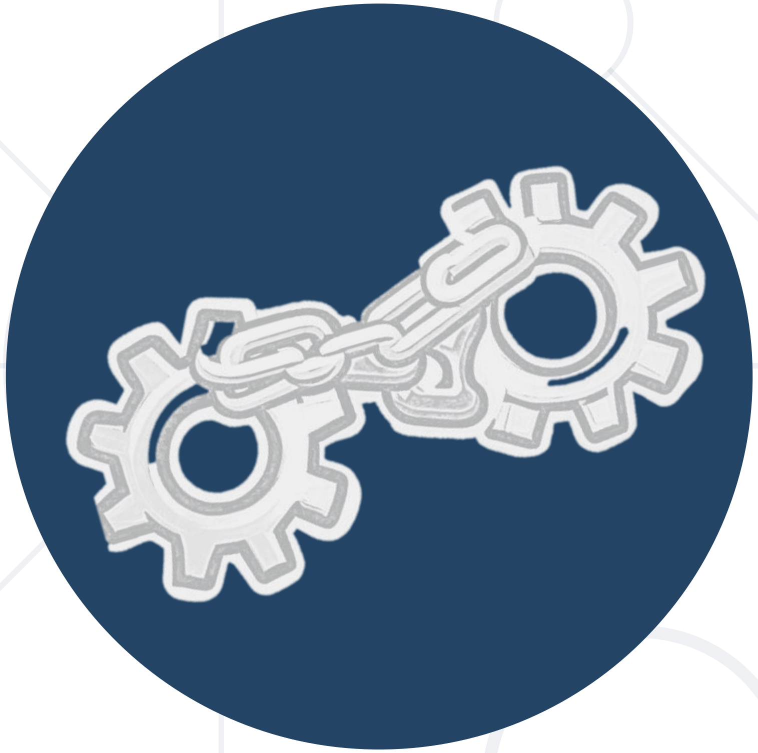
Solution: Delete Review by ID

```
# caller.py
def delete_review_by_id(review_id):
    review_to_delete = Review.objects.get(pk=review_id)

    review_to_delete.delete()

    return f"{str(review_to_delete)} was deleted"
```

Review model __str__ method



Chaining Methods

Chaining Methods

- Chaining **methods** in Django
  - A **powerful** way to construct **complex queries**
    - applying **multiple operations** on a **QuerySet** in a **single line** of code
  - Performs operations on the **resulting QuerySet** in an **expressive** and **readable** manner
  - Allows building **flexible** and **dynamic queries**

Chaining Methods

- Most **methods** in Django's **QuerySet API** return a **new QuerySet** object
  - Enables **chaining filters**, **counting**, **ordering**, and other operations
- The **order** of the **chained methods matters**
  - **Each** method **operates** on the **QuerySet returned** by the **previous** method
  - You need to **consider** the **logical order** of the operations

Chaining Methods

- Using **multiple methods** in a **single** query

caller.py

```
def count_seniors():
    number_of_sr_employees = 
Employee.objects.filter(job_level='Sr.').count()
    ...
```

First executes the filter, then counts the filtered records

Retrieve the number of senior employees

Problem: Filter Authors by Nationalities

- Create a function called **`"filter_authors_by_nationalities"`** that:
  - **Receives a nationality** as an argument
  - **Filters** only the **authors** with the **given nationality** and **orders them by first name**, and then by **last name**
  - **Returns information** about each found **author's biography** in the format:

    ```
    "{biography1}"
    ...
    "{biographyN}"
    ```

    - If there is **NO biography added** for an author, return **information about their full name** in the format: **`"{first_name} {last_name}"`**

Solution: Filter Authors by Nationalities

```python
# caller.py
def filter_authors_by_nationalities(nationality):
    filtered_authors = (Author.objects
                        .filter(nationality=nationality)
                        .order_by("first_name", "last_name"))

    result = [a.biography
              if a.biography is not None
              else f"{a.first_name} {a.last_name}"
              for a in filtered_authors]

    return "\n".join(result)
```

Lookup Keys

Lookup Keys in Django

- Used in query operations to **specify conditions** or **filters** on the **fields** of a model
- Used in **conjunction** with the query **methods** such as **`filter()`**, **`exclude()`**, and **`get()`** to perform **precise** database **queries**
- Added to the **field names** in the query to **define** the **type** of **comparison** or **operation** to be performed on the **field values**
- A way to **specify** how the **values** of the **fields** should be **compared** or **matched** against the provided query **criteria**

Lookup Keys

- Using the format: **`field__lookupkey=value`**:

caller.py

```
def get_employees_id_lte():
    employees = Employee.objects.filter(id__lte=5)
    ...
```

field

lookup key

value

Returns a QuerySet with all employees (objects) whose id is less than or equal to 5

Lookup Keys

- Matching the **exact value** of the field (**case-sensitive** by **default**)

```
Employee.objects.filter(job_level="Jr.")
Employee.objects.exclude(job_level__exact="Jr.")   # explicit form
Employee.objects.get(email_address__iexact="a@b.com") # case-insensitive match
```

- Matching **values** that **contain** a **specific substring**

```
Employee.objects.exclude(job_title__contains="Engineer")
Employee.objects.filter(job_title__icontains="engineer") # case-insensitive
```

- Matching **values starting with** or **ending with** a given **string**

```
Employee.objects.exclude(job_level__startswith="Sr.")
Employee.objects.filter(job_title__endswith="Engineer")
```

Lookup Keys

- Matching field values **greater than** a given value

```
Employee.objects.filter(id__gt=2)    # greater than
Employee.objects.exclude(id__gte=2)  # greater than or equal to
```

- Matching field values **less than** a given value

```
Employee.objects.filter(id__lt=5)    # less than
Employee.objects.exclude(id__lte=5) # less than or equal to
```

- Matching field values **in a range** (inclusive)

```
Employee.objects.filter(id__range=(2, 5)) # from 2 to 5, both inclusive
```

Additional Field Lookups

- **Date/time** field allows **chaining additional field** lookups

caller.py

```
def get_employees_by_bd_year():
    employees = Employee.objects.filter(birth_date__year__gt=1999)
    ...
```

field

additional field

lookup key

Returns a QuerySet with all employees (objects) who are born after 1999

More at: https://docs.djangoproject.com/en/5.0/ref/models/querysets/#date

Problem: Filter Authors by Birth Year

- Create a function called "`filter_authors_by_birth_year`":
  - It **receives two years** as two arguments
  - **Filters** the **authors who are born between the two given years (both inclusive)** and **order them by birth date** in **descending order**
  - **Returns information** about each found **author** in the format:

```
"{birth_date}: {first_name} {last_name}"
...
"{birth_date}: {first_name} {last_name}"
```

Solution: Filter Authors by Birth Year

```
# caller.py
def filter_authors_by_birth_year(first_year, second_year):
    filtered_authors = (Author.objects.filter(
        birth_date__year__range=(first_year, second_year))
                            .order_by("-birth_date"))

    result = [f"{a.birth_date}: {a.first_name} {a.last_name}"
              for a in filtered_authors ]

    return "\n".join(result)
```

Bulk Methods

Bulk Create, Update, Delete

Bulk Methods

- Used to **perform** database **operations** **efficiently**
  - **on multiple objects simultaneously**
  - **instead of individually** processing **each object**
- Provide a way to **optimize** database **interactions**
- Improve **performance**
  - when dealing with **a large number** of **objects**

Using Bulk Methods

- **bulk_create()** method
  - Creates **multiple objects** in a **single** database **query**
  - Accepts a **list** of **object instances** as an argument
    - **efficiently inserts** them into the database

caller.py

```
def bulk_create_employees():
    new_employees = [
        Employee(…), Employee(…), Employee(…)…
    ]
    bulk_employees = Employee.objects.bulk_create(new_employees)
    ...
```

List of object instances

Creating and saving all new records at once

Using Bulk Methods

- **Bulk update**
  - Chain `filter()` and `update()` methods

caller.py

```
def bulk_update_employees():
    updated_employees =
Employee.objects.filter(job_level='Jr.').update(job_level='Mid')
    ...
```

Filtering with condition

New value

Updating all filtered records at once

Using Bulk Methods

- **Bulk delete**
  - Chain `filter()` and `delete()` methods

caller.py

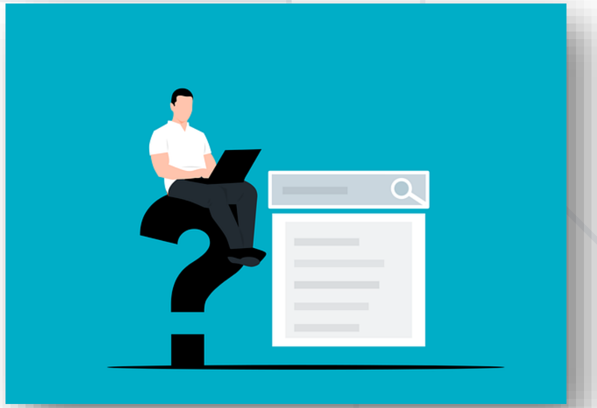
```
def bulk_delete_employees():
    deleted_employees =
Employee.objects.filter(job_level='Mid').delete()
    ...
```

Filtering with condition

Deleting all filtered records at once

Problem: Change Reviewer's Name

- Create a function called "`change_reviewer_name`" that:
  - Receives the **reviewer's name** as a first argument and a **new name** as a second argument
  - Changes **all occurrences** of the **reviewer's name** with the **new name**
  - **Returns** a queryset of all **review records**



Solution: Change Reviewer's Name

```python
# caller.py
def change_reviewer_name(reviewer_name, new_name):
    (Review.objects
     .filter(reviewer_name=reviewer_name)
     .update(reviewer_name=new_name))

    result = Review.objects.all()

    return result
```

Summary

- Useful Methods
  - `filter()`, `count()`, `get()`
- Lookup Keys
  - `iexact`, `icontains`
- Bulk Methods
  - `bulk_create()`

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

СофтУни БУДИТЕЛ

Finance Academy

SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

SOFTWARE GROUP

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License