Data Operations in Django with Queries







SoftUni Team Technical Trainers







https://softuni.bg

Table of Contents



- 1. CRUD Operations
 - Introduction
 - Importance
- 2. Query Set
- 3. Django Simple Queries
 - Default Object Manager
 - Create, Select, Update, and Delete
- 4. Django Shell and SQL Logging



Have a Question?



sli.do

#python-db



CRUD Overview

Importance of CRUD

What is CRUD?



- An acronym that stands for
 - Create, Read, Update, and Delete
- A set of four basic operations used in
 - Web development
 - Database management
- Represents the fundamental actions, performed on data stored in a database



CRUD – Four Fundamental Actions



Create

Creation of new data records/entities in the database

Read

Retrieve data from the database

Update

Modify or alter existing data records

Delete

Remove or delete data records from the database



CRUD Importance (1)



- The importance of CRUD operations lies in
 - their ability to provide a standardized and systematic way to interact with databases
- Forming the foundation of most web applications
- Vital for managing data throughout its lifecycle
- Developers can ensure that users could
 - create, retrieve, update, and delete data in a controlled and consistent manner
 - resulting in applications that are interactive, efficient, and capable of handling data manipulation tasks



CRUD Importance (2)



- Understanding CRUD operations is essential for
 - Anyone working with databases.
 - Web development
- They provide the basic building blocks for data management in modern applications



Django ORM and CRUD



- CRUD operations can be implemented using
 - Django's Object-Relational Mapping (ORM) capabilities
 - Providing a high-level API for interacting with databases
 - The ORM abstracts away the underlying databasespecific details
 - Allowing developers to focus on writing Python code
 - Performing CRUD operations on database models





QuerySet (1)



- A powerful feature that allows you to
 - Retrieve, filter, and manipulate data from the database using Python code
- Acts as an intermediary between your Python code and the database
- When performing a query on a model, Django returns
 - A QuerySet object that contains the results of the query



QuerySet (2)



- QuerySet contains a collection of objects from the database
- Doesn't directly fetch the data from the database
 - until explicitly evaluate or access its data

```
caller.py

def show_all_employees():
    all_employees = Employee.objects.all()
    print(all_employees)
    # <QuerySet []>
    print(type(all_employees))
    # <class 'django.db.models.query.QuerySet'>
```



QuerySet Key Features



- Lazy Evaluation
- Retrieving Objects
- Chaining Filters
- Querying Related Objects
- Aggregation and Annotation
- Ordering
- Pagination





Django Simple Queries

Create, Read, Update, and Delete

Object Manager



- Object Manager is a class attribute of a model
 - Provides methods for querying the database
- The default object manager is named objects
 - Automatically created for every model
 - Allowing to perform common database operations
 - filtering, retrieving all objects, getting a single object, etc.
- You can also define custom object managers in your models



Common Object Manager Methods



- all()
 - returns a query set containing all objects created so far
- first()
 - returns the first record from the DB
- get(**kwargs)
 - returns a single object that matches the given argument
 - If multiple objects are found it will throw a Model.MultipleObjectsReturned error
 - If get() doesn't find any object, it raises a Model.DoesNotExist exception
- create(**kwargs)
 - creates a new object with given arguments



Common Object Manager Methods (2)



- filter(**kwargs)
 - returns a query set containing a list of objects that match the given arguments
- exclude(**kwargs)
 - it does exactly the opposite of filter() method i.e.
 returns a queryset containing objects that do not match
 the given arguments
- order_by(*fields)
 - sets the ordering of the previously returned queryset according to the arguments passed in it



Creating Records in Database (1)



- Use Python code to create records in a database table
 - Use the object manager's method create()
 - Use it to create and save an object in a single step

```
caller.py

def create_employee():
    Employee.objects.create(first_name='Paul', last_name='Smith',
    job_title= ...)
```

Default object manager

Creates and saves a record with the given arguments

Creating Records in Database (2)



Use Python code to create records in a database table

Saving the object into the DB

- Instantiate a model object
- Call the save() method on it

Problem: Add Students



- You are given an ORM project skeleton (you can download it from here) with one model called "Student"
- Add these records to the students' database table

student_id	first_name	last_name	birth_date	email
FC5204	John	Doe	15/05/1995	john.doe@university.com
FE0054	Jane	Smith	null	jane.smith@university.com
FH2014	Alice	Johnson	10/02/1998	alice.johnson@university.com
FH2015	Bob	Wilson	25/11/1996	bob.wilson@university.com

Solution: Add Students

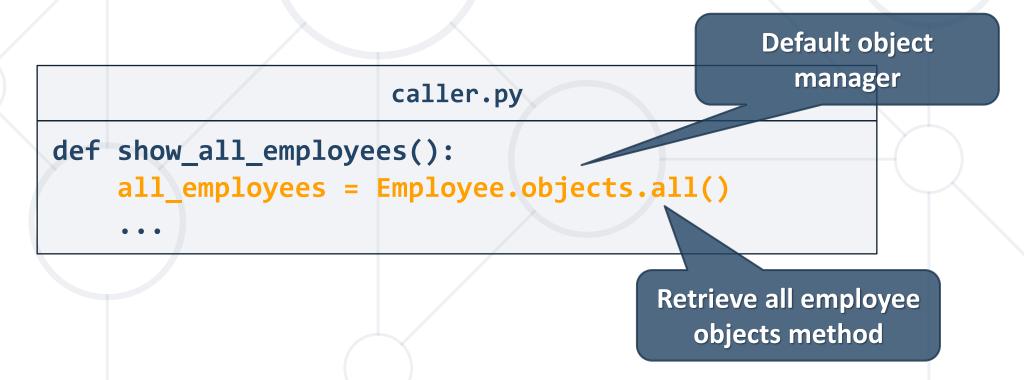


```
def add_students():
    Student.objects.create(
        student_id="FC5204",
        first name="John",
        last name="Doe",
        birth date="1995-05-15",
        email="john.doe@university.com")
    student = Student(
        student id="FE0054",
        first_name="Jane",
        last name="Smith",
        email="jane.smith@university.com")
    student.save()
```

Reading Data from the Database



- Use Python code to retrieve data from the database
 - Use the Default Object Manager on the model class
 - Make a Django query that returns a QuerySet containing all objects



Problem: Get Students Info



Create a function called "get_students" that returns all students' records from the database in the format:

"Student No{student_id}: {first_name} {last_name}; Email: {email}"



Solution: Get Students Info



```
def get_students_info():
    students = Student.objects.all()
    students_info = []
    for student in students:
        students info.append(
            f"Student №{student.student_id}:
            f"{student.first_name} {student.last_name}; "
            f"Email: {student.email}"
    return "\n".join(students_info)
```

Updating a Record in Database



- Use Python code to update a record from the database table
 - Retrieve the object using a query
 - Modify its attributes
 - Save it

```
caller.py

def update_first_employee():
    first_employee = Employee.objects.first()
    first_employee.job_level = 'Sr.'
    first_employee.save()
Retrieving the first record
```

Saving the updated object into the DB

Problem: Update Students' Emails



- Create a function called "update_students_emails" that updates all students' emails
 - The current email domain is "university.com"
 - You need to change it to "uni-students.com"



Solution: Update Students' Emails





Deleting a Record from Database



- Immediately delete the object/s from the database
- You should explicitly request the object
- The method returns:
 - The number of objects deleted
 - The number of deletions per object type

```
employee = Employee.objects.first()
result = employee.delete()
...
# (1, {'main_app.Employee': 1})
```

Deleting Model Object (2)



Delete a single object

```
employee = Employee.objects.first()
employee.delete()
```

Delete multiple objects in a QuerySet

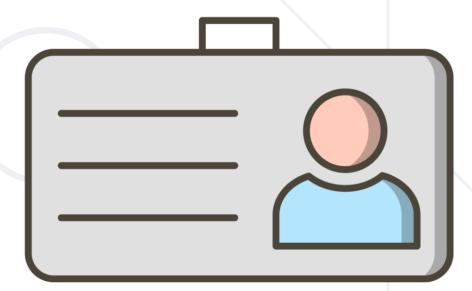
```
employees = Employee.objects.all()
employees.delete()
```

 Note: When deleting an object with foreign keys, it will follow the behavior of the SQL constraint ON DELETE

Problem: Truncate Students



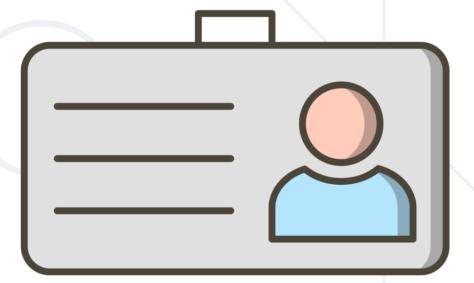
Create a function called "truncate_students" that deletes
 all student's records from the database table



Solution: Truncate Students



```
def truncate_students():
    students = Student.objects.all()
    students.delete()
```





Django Shell & SQL Logging

Django Shell (1)



- Django shell is a command-line interface
 - Allows you to interact with your Django project
 - Uses Python code
 - Provides a convenient way to
 - Experiment
 - Test
 - Debug



Django Shell Usage



Run the following command in your project's directory

```
python manage.py shell
```

Interact with models and perform database operations

```
# Import the necessary models
from main_app.models import Employee

# Retrieve all records
all_employees = Employee.objects.all()
```

SQL Logging in Django



- Capturing and viewing the SQL queries executed by Django when interacting with the database
- Useful for debugging
- Optimize database-related operations
- Provides details about
 - The executed SQL statement
 - Execution time
 - Other relevant information
- Invaluable in understanding and optimizing database performance



SQL Logging Configuration



 Add the following configuration to your settings.py file to enable SQL logging

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler'}},
    'root': {
        'handlers': ['console'],
        'level': 'DEBUG',
    'loggers': {
        'django.db.backends': {
            'handlers': ['console'],
            'level': 'DEBUG',
            'propagate': False,
        }}}
```

SQL Logging - View Results



SQL Queries will be logged on the console





Live Demo

SQL Logging

Summary



- CRUD
- QuerySet
- Django Simple Queries
 - Default Object Manager
- Django Shell
- SQL Logging





Questions?

















SoftUni Diamond Partners



SUPER HOSTING .BG



























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
- Software University Forums
 - forum.softuni.bg









License



- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is copyrighted content
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni https://softuni.org
- © Software University https://softuni.bg

