TRAINING DAY9 REPORT

03 JULY 2025

CRUD OPERATIONS

CRUD stands for Create, Read, Update, Delete — these are the basic operations for managing data in any application.

1. **Create**

To create a record in the database, you can use a form or directly use the ORM.

Model:

```
blogapi > blog > models.py > ...
1   from django.db import models
2
3   from django.db import models
4
5   class Product(models.Model):
6       name = models.CharField(max_length=100)
7       price = models.DecimalField(max_digits=8, decimal_places=2)
8       description = models.TextField()
9
```

View using CreateView (Class-Based View):

```
blogapi > blog > views.py > ...
1   from django.views.generic.edit import CreateView
2   from .models import Product
3
4   class ProductCreateView(CreateView):
5       model = Product
6       fields = ['name', 'price', 'description']
7       template_name = 'product_form.html'
8       success_url = '/products/'
```

2. **Read**

To display or list items:

View using ListView and DetailView:

```
blogapi > blog > views.py > ...
from django.views.generic import ListView, DetailView

class ProductListView(ListView):
    model = Product
    template_name = 'product_list.html'

class ProductDetailView(DetailView):
    model = Product
    template_name = 'product_detail.html'

```

3. Update

To update existing records:

UpdateView:

```
blogapi > blog > views.py > ...
from django.views.generic.edit import UpdateView

class ProductUpdateView(UpdateView):
    model = Product
    fields = ['name', 'price', 'description']
    template_name = 'product_form.html'
    success_url = '/products/'

```

4. Delete

To delete records:

DeleteView:

```
blogapi > blog > views.py > ...
from django.views.generic.edit import DeleteView
from django.urls import reverse_lazy

class ProductDeleteView(DeleteView):
    model = Product
    template_name = 'product_confirm_delete.html'
    success_url = reverse_lazy('product-list')

```

WHAT IS A QUERYSET?

A **QuerySet** is a collection of database queries to retrieve data from the database in Django. It is **lazy**, meaning it only hits the database when evaluated.

Common QuerySet Methods

Our model:

blogapi > blog > models.py > ...

```python
from django.db import models

class Product(models.Model):
    name = models.CharField(max_length=100)
    category = models.CharField(max_length=50)
    price = models.DecimalField(max_digits=8, decimal_places=2)
    in_stock = models.BooleanField(default=True)
    created_at = models.DateTimeField(auto_now_add=True)

```

Querysets --

1. all()

blogapi > blog > views.py > ...

```python
from django.shortcuts import render
from .models import Product

def products(request):
    products = Product.objects.all()
    return render(request, 'product_list.html', {'products': products})

```

2. filter()

```
blogapi > blog > views.py > ...
1  from django.shortcuts import render
2  from .models import Product
3
4  def products(request):
5      products = Product.objects.filter(price__lte=1000)
6      return render(request, 'product_list.html', {'products': products})
7
```

3. get()

```
blogapi > blog > views.py > ...
1  from django.shortcuts import render
2  from .models import Product
3
4  def products(request):
5      products = Product.objects.get(id=1)
6      return render(request, 'product_list.html', {'products': products})
7
8
```

4. values()

```
blogapi > blog > views.py > ...
1  from django.shortcuts import render
2  from .models import Product
3
4  def products(request):
5      products = Product.objects.filter(in_stock=True).values('name', 'price')
6      return render(request, 'product_list.html', {'products': products})
7
```

5. count()

```
blogapi > blog > views.py > ...
1  from django.shortcuts import render
2  from .models import Product
3
4  def products(request):
5      products = Product.objects.filter(in_stock=True).count()
6      return render(request, 'product_list.html', {'products': products})
7
```

6. first()

```
blogapi > blog > views.py > ...
1   from django.shortcuts import render
2   from .models import Product
3
4   def products(request):
5       products = Product.objects.order_by('created_at').first()
6       return render(request, 'product_list.html', {'products': products})
7
8
```

7. update()

```
blogapi > blog > views.py > ...
1   from django.shortcuts import render
2   from .models import Product
3
4   def products(request):
5       products = Product.objects.filter(category="Shoes").update(price=500)
6       return render(request, 'product_list.html', {'products': products})
7
8
```

8. delete()

```
blogapi > blog > views.py > ...
1   from django.shortcuts import render
2   from .models import Product
3
4   def products(request):
5       products = Product.objects.filter(in_stock=False).delete()
6       return render(request, 'product_list.html', {'products': products})
7
```

9. create()

```
blogapi > blog > views.py > ...
1   from django.shortcuts import render
2   from .models import Product
3
4   def products(request):
5       products = Product.objects.create(
6           name='Watch',
7           category='Accessories',
8           price=1500)
9       return render(request, 'product_list.html', {'products': products})
10
```

10. reverse()

```
blogapi > blog > views.py > ...
1   from django.shortcuts import render
2   from .models import Product
3
4   def products(request):
5       products = Product.objects.all().order_by('name').reverse()
6       return render(request, 'product_list.html', {'products': products})
7
8
```