**Model Definition**

|  |
| --- |
| **from django.db import models**  **class MyModel(models.Model):**  **field\_name = models.CharField(max\_length=100)**  **another\_field = models.IntegerField()**  **date\_field = models.DateTimeField(auto\_now\_add=True)** |

**Queryset Basics**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CRUD Operations & Queries**  **Create Objects:**   |  | | --- | | **from myapp.models import Product**  **# Create a new product**  **product1 = Product(name='Laptop', price=999.99, quantity=5, description='High-performance laptop')**  **product1.save()**  **# Alternative: Create and save in one step**  **product2 = Product.objects.create(name='Smartphone', price=499.50, quantity=10,**  **description='Latest smartphone model')** |     **Retrieve Objects:**   |  | | --- | | **# Get all products**  **all\_products = Product.objects.all()**  **# Get a specific product by its primary key (ID)**  **laptop = Product.objects.get(pk=1)**  **# Filter products by a condition**  **laptops = Product.objects.filter(name\_\_contains='Laptop') # Products with 'Laptop' in the name**  **# Chaining filters**  **affordable\_laptops = Product.objects.filter(name\_\_contains='Laptop').filter(price\_\_lte=800)** |   **Update Objects:**   |  | | --- | | **# Update a product's price**  **laptop = Product.objects.get(pk=1)**  **laptop.price = 949.99**  **laptop.save()**  **# Update multiple products at once**  **Product.objects.filter(name\_\_contains='Laptop').update(quantity=10)** |   **Delete Objects:**   |  | | --- | | **# Delete a specific product**  **laptop = Product.objects.get(pk=1)**  **laptop.delete()**  **# Delete multiple products**  **Product.objects.filter(quantity=0).delete()** | |  | | **# Count the number of products**  **total\_products = Product.objects.count()**  **# Check if a product exists**  **has\_laptops = Product.objects.filter(name\_\_contains='Laptop').exists()**  **# Exclude products**  **non\_laptop\_products = Product.objects.exclude(name\_\_contains='Laptop')**  **# Order products**  **products\_by\_price = Product.objects.order\_by('price')** | |

**Creating and Updating Objects**

|  |
| --- |
| **# Create and Save**  **obj = MyModel(field\_name='value', another\_field=42)**  **obj.save()**  **# Bulk Create (improves performance)**  **MyModel.objects.bulk\_create([**  **MyModel(field\_name='value1', another\_field=42),**  **MyModel(field\_name='value2', another\_field=43),**  **])**  **# Update**  **MyModel.objects.filter(field\_name='old\_value').update(field\_name='new\_value')** |

**Querying with Q Objects (Complex Queries)**

|  |
| --- |
| **from django.db.models import Q**  **# OR query**  **q = Q(field\_name='value') | Q(another\_field=42)**  **or\_query = MyModel.objects.filter(q)**  **# AND query**  **q = Q(field\_name='value') & Q(another\_field=42)**  **and\_query = MyModel.objects.filter(q)** |

**Related Objects**

|  |
| --- |
| **from myapp.models import MyModel, RelatedModel**  **# One-to-One Relationship**  **class MyModel(models.Model):**  **related\_model = models.OneToOneField(RelatedModel, on\_delete=models.CASCADE)**  **# One-to-Many Relationship**  **class MyModel(models.Model):**  **related\_models = models.ForeignKey(RelatedModel, on\_delete=models.CASCADE)**  **# Many-to-Many Relationship**  **class MyModel(models.Model):**  **related\_models = models.ManyToManyField(RelatedModel)** |

**Related Objects Querying**

|  |
| --- |
| **# Reverse relation**  **related\_objects = RelatedModel.objects.filter(mymodel\_\_field\_name='value')**  **# Prefetch related objects (reduce queries)**  **my\_objects = MyModel.objects.prefetch\_related('related\_models')** |

**Aggregation and Annotation**

|  |
| --- |
| **from django.db.models import Avg, Sum, Count**  **# Aggregate functions**  **average\_value = MyModel.objects.aggregate(avg=Avg('another\_field'))**  **total\_sum = MyModel.objects.aggregate(sum=Sum('another\_field'))**  **total\_count = MyModel.objects.aggregate(count=Count('pk'))**  **# Annotate (add calculated fields)**  **annotated\_objects = MyModel.objects.annotate(avg=Avg('another\_field'))** |

## F-Expressions (Update and Annotate)

|  |
| --- |
| **from django.db.models import F**  **# Update fields with F-expression**  **MyModel.objects.update(another\_field=F('another\_field') + 10)**  **# Annotate with F-expression**  **annotated\_objects = MyModel.objects.annotate(sum=F('another\_field') + F('field\_name'))** |

## Transactions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **from django.db import transaction**  **# Manual transaction**  **with transaction.atomic():**  **# Your transactional operations here**   |  | | --- | | **Example 1: Transferring Funds Between Accounts**  **from django.db import transaction**  **@transaction.atomic**  **def transfer\_funds(sender\_account\_id, receiver\_account\_id, amount):**  **sender\_account = Account.objects.get(pk=sender\_account\_id)**  **receiver\_account = Account.objects.get(pk=receiver\_account\_id)**  **if sender\_account.balance >= amount:**  **sender\_account.balance -= amount**  **receiver\_account.balance += amount**  **sender\_account.save()**  **receiver\_account.save()**  **else:**  **raise ValueError("Insufficient funds")**   1. **Atomicity: The transfer is atomic. Either both accounts are updated, or neither is (in case of insufficient funds).** 2. **Consistency: Account balances remain consistent. No money is lost or created.** | | **Example 2: Creating a Blog Post with Related Tags**  **Python**  **@transaction.atomic**  **def create\_blog\_post(title, content, tag\_names):**  **post = BlogPost.objects.create(title=title, content=content)**  **for tag\_name in tag\_names:**  **tag, created = Tag.objects.get\_or\_create(name=tag\_name)**  **post.tags.add(tag)**   * **Atomicity: If tag creation fails for any reason, the blog post won't be created.** * **Consistency: The blog post is always associated with the correct tags.** | |  | | **Example 3: Enrolling Students in a Course (with Limited Capacity)**  **Python**  **@transaction.atomic**  **def enroll\_student(course\_id, student\_id):**  **course = Course.objects.select\_for\_update().get(pk=course\_id)**  **if course.enrollment\_count < course.capacity:**  **course.enrollment\_count += 1**  **course.save()**  **Enrollment.objects.create(course=course, student\_id=student\_id)**  **else:**  **raise ValueError("Course is full")**   1. **select\_for\_update()**: **Locks the row to prevent race conditions where multiple users might try to enroll simultaneously in a course with limited capacity.** | |

## Signals

|  |
| --- |
| **Example 1: Sending a Notification Email**  **Python**  **from django.core.mail import send\_mail**  **from django.conf import settings # For email settings**  **from myapp.models import Order**  **@receiver(post\_save, sender=Order)**  **def order\_created\_notification(sender, instance, created, \*\*kwargs):**  **if created: # Only for new orders**  **subject = f"Order #{instance.id} Created"**  **message = f"Thank you for your order!\nYour order details: {instance}"**  **send\_mail(subject, message, settings.DEFAULT\_FROM\_EMAIL, [instance.customer.email])** |
| **Example 2: Updating Related Data**  **Python**  **from myapp.models import Product, Inventory**  **@receiver(post\_save, sender=Product)**  **def update\_inventory(sender, instance, \*\*kwargs):**  **inventory, created = Inventory.objects.get\_or\_create(product=instance)**  **inventory.quantity = instance.quantity # Sync inventory with product quantity**  **inventory.save()** |
| **Example 3: Creating User Profiles**  **Python**  **from django.contrib.auth.models import User**  **from myapp.models import UserProfile**  **@receiver(post\_save, sender=User)**  **def create\_user\_profile(sender, instance, created, \*\*kwargs):**  **if created:**  **UserProfile.objects.create(user=instance)** |

Filtering with Lookups

|  |
| --- |
| **# Case-insensitive exact match**  **filtered\_objects = MyModel.objects.filter(field\_name\_\_iexact='value')**  **# Contains**  **filtered\_objects = MyModel.objects.filter(field\_name\_\_contains='value')**  **# Startswith and Endswith**  **filtered\_objects = MyModel.objects.filter(field\_name\_\_startswith='prefix')**  **filtered\_objects = MyModel.objects.filter(field\_name\_\_endswith='suffix')**  **# In**  **filtered\_objects = MyModel.objects.filter(another\_field\_\_in=[1, 2, 3])**  **# Range**  **filtered\_objects = MyModel.objects.filter(another\_field\_\_range=(10, 20))** |

## Date and Time Queries

|  |
| --- |
| from datetime import date  # Exact Date Match  filtered\_objects = MyModel.objects.filter(date\_field\_\_date=date(2023, 7, 31))  # Year, Month, Day  filtered\_objects = MyModel.objects.filter(date\_field\_\_year=2023)  filtered\_objects = MyModel.objects.filter(date\_field\_\_month=7)  filtered\_objects = MyModel.objects.filter(date\_field\_\_day=31)  # Greater Than and Less Than  filtered\_objects = MyModel.objects.filter(date\_field\_\_gt=date(2023, 7, 1))  filtered\_objects = MyModel.objects.filter(date\_field\_\_lt=date(2023, 8, 1)) |

## Working with Aggregates and Grouping

|  |
| --- |
| **1. Calculate Total Sales:**  **Python**  **from django.db.models import Sum**  **from myapp.models import Order**  **total\_sales = Order.objects.aggregate(Sum('total\_price'))**  **print(total\_sales['total\_price\_\_sum']) # Output: e.g., 12345.67** |
| **2. Average Order Value:**  **Python**  **from django.db.models import Avg**  **average\_order\_value = Order.objects.aggregate(Avg('total\_price'))**  **print(average\_order\_value['total\_price\_\_avg']) # Output: e.g., 56.78** |
| **3. Count of Orders per Customer:**  **Python**  **from django.db.models import Count**  **orders\_per\_customer = Order.objects.values('customer').annotate(order\_count=Count('id'))**  **for item in orders\_per\_customer:**  **print(f"Customer: {item['customer']}, Orders: {item['order\_count']}")** |
| **4. Top-Selling Products:**  **Python**  **from django.db.models import Sum**  **top\_products = Product.objects.annotate(total\_sold=Sum('orderitem\_\_quantity')).order\_by('-total\_sold')[:5]**  **for product in top\_products:**  **print(f"{product.name}: {product.total\_sold} sold")** |
| **5. Monthly Sales Report:**  **Python**  **from django.db.models import Sum**  **from django.db.models.functions import TruncMonth**  **monthly\_sales = Order.objects.annotate(month=TruncMonth('created\_at')).values('month').annotate(total\_sales=Sum('total\_price'))**  **for item in monthly\_sales:**  **print(f"Month: {item['month']}, Total Sales: {item['total\_sales']}")** |