 # Django Models and ORM (Object-Relational Mapping) - Complete Guide

---

🎓1. What are Models in Django?

In Django, **models** are Python classes that represent **database tables**. Each attribute in a model class represents a **column** in the table, and each instance of the class represents a **row**.

🔎Example:

```
from django.db import models

class Student(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    enrolled = models.BooleanField(default=True)
```

This creates a `Student` table with columns: `id`, `name`, `age`, `enrolled`

📊2. Why Use Models Instead of Raw SQL?

Models (ORM)	Raw SQL
Safer and less error-prone	Need to handle SQL manually
Database independent	Tied to specific SQL syntax
Automatically handles table creation	Must write SQL scripts
Easier to update and maintain	Harder to read/debug

Django models let you work with the database **like Python objects**, rather than writing SQL queries directly.

📃3. Common Field Types in Django Models

Field	Description
`CharField`	For small to medium strings (needs `max_length`)
`TextField`	For large text (no max_length required)
`IntegerField`	For whole numbers

Field	Description
`FloatField`	For decimal numbers
`BooleanField`	For True/False values
`DateField`	For dates
`DateTimeField`	For date and time
`EmailField`	Validates email format
`URLField`	Validates URLs
`FileField`	For uploading files
`ImageField`	For images (needs Pillow)
`ForeignKey`	One-to-many relationship
`ManyToManyField`	Many-to-many relationship
`OneToOneField`	One-to-one relationship

Example with different fields:

```
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    bio = models.TextField()
    birthdate = models.DateField()
    profile_pic = models.ImageField(upload_to='profiles/')
```

🕐4. What is ORM (Object-Relational Mapping)?

ORM is a technique that allows you to interact with the database using **Python code instead of SQL queries**.

Without ORM:

```
SELECT * FROM student WHERE age > 18;
```

With ORM:

```
Student.objects.filter(age__gt=18)
```

Benefits of ORM:

- Easier and readable syntax
- Automatic escaping (prevents SQL injection)
- Cross-database support
- Built-in validations

📚5. Using ORM in Views

Get All Records:

```
students = Student.objects.all()
```

Get First Record:

```
student = Student.objects.first()
```

Get Last Record:

```
student = Student.objects.last()
```

Filter Records:

```
adults = Student.objects.filter(age__gte=18)
```

Get a Single Record:

```
student = Student.objects.get(id=1)  # Raises error if not found
```

Safely Get a Record:

```
student = Student.objects.filter(id=1).first()  # Returns None if not found
```

Exclude:

```
non_adults = Student.objects.exclude(age__gte=18)
```

Count:

```
Student.objects.count()
```

Order By:

```
Student.objects.order_by('age')           # ascending
Student.objects.order_by('-age')          # descending
```

🕐6. Creating and Saving Records

```
new_student = Student(name="John", age=22)
new_student.save()
```

🔗7. Updating a Record

```
student = Student.objects.get(id=1)
student.age = 25
student.save()
```

🔩8. Deleting a Record

```
student = Student.objects.get(id=1)
student.delete()
```

📈9. Useful QuerySet Methods

Method	Description
`.all()`	Get all records
`.filter()`	Get filtered records
`.exclude()`	Get records NOT matching condition

Method	Description
`.get()`	Get single record (raises error if not found)
`.first()`	Get first object or None
`.last()`	Get last object or None
`.count()`	Total number of records
`.exists()`	Returns True/False if any record exists
`.order_by()`	Sort records

📚10. Summary

- Models define the structure of the database.
- Fields represent columns.
- ORM is the layer to interact with DB using Python.
- Django handles most DB operations automatically.
- Easy and safe to create, read, update, delete using ORM.

Let me know if you want a separate section on **relationships** (ForeignKey, ManyToMany, OneToOne) or **model methods** next!