Creating announcement model

1. from django.db import models
2. class Announcement(models.Model):
3. title = models.CharField(max\_length=200)
4. content = models.TextField()
5. date\_posted = models.DateTimeField(auto\_now\_add=True)
6. def \_\_str\_\_(self):
7. return self.title
8. we import models module for built in tools
9. By default:

**Class Name:** Announcement

**Table Name:** appname\_announcement and each row is an object

You are basically telling Django:

"Hey Django, I wanna make a **table** called announcement in the database. And this table will follow your rules (from models.Model)."

* models.Model ➔ The **parent class** that gives your class superpowers to behave like a table.
* Without models.Model, your class is just a plain Python class 😴.
* With it? It becomes a **Django model** = ready to create tables, save data, query data. 💪
* A table with 3 fields – title , content,data\_posted

1. A text field with a limit of **200 characters**.
2. A big text field — no length limit. You can dump paragraphs here.
3. This records the **date and time** automatically when a new announcement is created.
4. This method tells Django how to show this object as a string.
5. So when you print an Announcement object, you’ll see its **title** instead of some boring "Announcement object (1)"
6. **Visual Cheat Sheet: How This Maps to a Table**

| **Field Name** | **Data Type in Database** | **Example Value** |
| --- | --- | --- |
| id (Auto) | Auto-generated primary key | 1, 2, 3… |
| title | VARCHAR(200) | "College Fest Announcement" |
| content | TEXT | "All students are invited to..." |
| date\_posted | DATETIME | 2025-05-01 14:30:00 |

Save() in models

* "Hey Django, take this Python object and **save it into the database table**. Make it official!"
* It’s the **bridge** between:
  + Your Python code (models)
  + And your actual database (tables)
* If the object **doesn’t exist yet** ➔ .save() ➡️ **INSERT**
* If the object **already exists** ➔ .save() ➡️ **UPDATE**
* Python Object (Announcement)
* ↓
* .save() is called
* ↓
* DB INSERT or UPDATE command runs
* ↓
* Data saved permanently in SQL table
* a = Announcement(title="Hello", content="World")
* # Forgetting .save()
* Result ➡️ No data saved! 🤦‍♀️
* Always call ➡️ a.save() ➡️ Commit to DB!
* **1. .create() — Shortcut King!**
* Think of .create() as:
* **.save() + object creation** combined in one smooth move 💅
* **✅ Example:**
* python
* CopyEdit
* Announcement.objects.create(title="Holiday", content="Office will be closed tomorrow!")
* ➡️ This:
* Creates an Announcement object
* Calls .save() automatically
* Inserts into the DB **in one step**
* **🎯 Why use .create()?**

| * **When?** | * **Why?** |
| --- | --- |
| * Quick one-liners | * Less code, same effect |
| * Bulk scripts | * Fast and clean |

* **🌟 2. .save(commit=False) — The Sneaky Trickster**
* This one comes when you’re working with **forms** (like Django admin or user forms).  
  commit=False lets you:
* **Create the object ➡️ but wait before saving it.**  
  Useful when you wanna modify the object before saving.
* **✅ Example (Form Save Hack):**
* python
* CopyEdit
* form = AnnouncementForm(request.POST)
* if form.is\_valid():
* obj = form.save(commit=False) # 🚫 Not saved yet!
* obj.title = obj.title.upper() # 🔄 Modify before saving
* obj.save() # ✅ Now save!
* **🎯 Why use it?**

| * **When?** | * **Why?** |
| --- | --- |
| * Before saving, you wanna tweak data | * For validation or computed fields |
| * You have foreign keys to set manually | * Like assigning current user |

* **🌟 3. .bulk\_create() — Speed Booster 🚀**
* If you wanna save **many records at once**, looping .save() is slow. 🐢 Use .bulk\_create() ➡️ **Super fast batch insert** 🚀
* **✅ Example:**
* python
* CopyEdit
* Announcement.objects.bulk\_create([
* Announcement(title="News 1", content="Content 1"),
* Announcement(title="News 2", content="Content 2"),
* Announcement(title="News 3", content="Content 3"),
* ])
* ➡️ Saves 3 rows **in one go**!  
  (Django fires only **1 SQL query** instead of 3 separate .save() calls!)
* **🎯 Why use it?**

| * **When?** | * **Why?** |
| --- | --- |
| * Inserting 100s/1000s rows | * Super fast! Reduces DB hits |
| * Data import scripts | * Performance-friendly |

* **🌟 4. .update() — Direct Table Edit**
* This skips objects and hits the table directly.  
  (No .save(), no signals ➡️ raw & fast)
* **✅ Example:**
* python
* CopyEdit
* Announcement.objects.filter(title="Old News").update(content="Updated Content!")
* ➡️ Directly updates in the DB table.
* **🎯 Why use it?**

| * **When?** | * **Why?** |
| --- | --- |
| * Mass updates | * Fast, avoids object loading |
| * No need for .save() | * Performance boost |

* **🕶️ Ultimate Cheat Table**

| * **Method** | * **Purpose** | * **Speed** | * **Saves signals?** |
| --- | --- | --- | --- |
| * .save() | * Save 1 object | * Normal | * ✅ Yes |
| * .create() | * Create & save 1 object (shortcut) | * Normal | * ✅ Yes |
| * .bulk\_create() | * Save multiple objects fast | * 🚀 Super Fast | * ❌ No signals |
| * .update() | * Update rows directly | * 🚀 Super Fast | * ❌ No signals |
| * .save(commit=False) | * Delay saving (edit first) | * Normal | * ✅ Yes |

* **🎯 Pro Tip for Interviews / Real Projects**
* ✅ .save() ➡️ Use when dealing with **single objects** and want signals/triggers.
* ✅ .bulk\_create() & .update() ➡️ Use when dealing with **large data sets** and care about **speed**.
* **💥 Visual Graph: Model Saving Options**
* sql
* CopyEdit
* Single Save ➡️ .save() / .create()
* ↓
* Bulk Save ➡️ .bulk\_create()
* ↓
* Bulk Update ➡️ .update()
* **🏅 Your Ninja Powers Now:**
* .save() ➡️ Regular saving ✅
* .create() ➡️ Quick & clean ✅
* .bulk\_create() ➡️ Speed for bulk ✅
* .update() ➡️ Direct & efficient ✅
* .save(commit=False) ➡️ Sneaky edits ✅
* **What are Models in Django?**
* Models = 💽 **Database tables in Python code**  
  They define the structure of your data: fields, types, relationships.
* **Think:**
* Model ➔ Table 🪑
* Field ➔ Column 📊
* Object ➔ Row 📄
* In Django:
* "Models are the single source of truth for your database structure." 🔥
* **🛠️ Basic Structure of a Model**
* python
* CopyEdit
* from django.db import models
* class MyModel(models.Model):
* field1 = models.CharField(max\_length=100)
* field2 = models.IntegerField()

| * **Part** | * **Meaning** |
| --- | --- |
| * class | * Defines the table structure. |
| * models.Model | * Makes it a Django model. |
| * field1 | * This becomes a table column. |

* **📦 Common Field Types (Memorize like ABCs)**

| * **Field Type** | * **Use For** | * **Example** |
| --- | --- | --- |
| * CharField | * Short text (needs max\_length) | * Name, Title |
| * TextField | * Long text | * Description, Content |
| * IntegerField | * Whole numbers | * Age, Count |
| * FloatField | * Decimals | * Price, Ratings |
| * BooleanField | * True / False | * Is Active? |
| * DateTimeField | * Date + Time | * Created At |
| * DateField | * Only Date | * Birthdate |
| * TimeField | * Only Time | * Event time |
| * EmailField | * Email validation | * User Email |
| * URLField | * Web URLs | * Profile Link |
| * FileField | * File upload | * Resume, Documents |
| * ImageField | * Image upload | * Profile Pic, Banner |

* **🔗 Relationships (Real Power 💪)**

| * **Field Type** | * **Meaning** | * **Example** |
| --- | --- | --- |
| * ForeignKey | * Many-to-One | * Post ➔ belongs to ➔ Author |
| * OneToOneField | * One-to-One | * User ➔ has ➔ Profile |
| * ManyToManyField | * Many-to-Many | * Students ➔ enrolled in ➔ Courses |

* **📊 Visual Cheat Sheet**
* 🧑 ➔ 📝 = ForeignKey (Author ➔ Posts)
* 🧑 ➔ 👤 = OneToOneField (User ➔ Profile)
* 🧑➕🧑 ➔ 📚 = ManyToManyField (Students ➔ Courses)
* **🚀 Model Options (Meta class ✨)**

| * **Option** | * **Use For** | * **Example** |
| --- | --- | --- |
| * ordering | * Default order of records | * ordering = ['-created\_at'] |
| * verbose\_name | * Human-readable singular name | * verbose\_name = "Blog Post" |
| * db\_table | * Custom table name | * db\_table = "my\_custom\_table" |

* python
* CopyEdit
* class Meta:
* ordering = ['-date\_posted']
* verbose\_name = "Announcement"
* **💣 Special Fields (Auto Work Done By Django 🪄)**

| * **Field Option** | * **Effect** | * **Example** |
| --- | --- | --- |
| * auto\_now\_add=True | * Set timestamp when created | * created\_at = models.DateTimeField(auto\_now\_add=True) |
| * auto\_now=True | * Set timestamp every time updated | * updated\_at = models.DateTimeField(auto\_now=True) |
| * unique=True | * No duplicates allowed | * username = models.CharField(unique=True) |
| * null=True | * Field can be NULL in DB |  |
| * blank=True | * Field can be blank in forms |  |

* **🏹 Django Model Methods (Boss Level 🕶️)**

| * **Method Type** | * **Purpose** | * **Example** |
| --- | --- | --- |
| * \_\_str\_\_() | * String representation | * return self.title |
| * save() | * Save record | * .save() |
| * delete() | * Delete record | * .delete() |
| * get\_absolute\_url() | * URL for object detail view | * reverse('post-detail', args=[str(self.id)]) |

* **👑 Model Lifecycle in Real Projects**

| * **Step** | * **Command** | * **Meaning** |
| --- | --- | --- |
| * Define model | * Write in models.py | * Table structure ready |
| * Make migration | * python manage.py makemigrations | * Create migration file (blueprint) |
| * Apply migration | * python manage.py migrate | * Actually create table in DB 🪄 |
| * Use model | * In views, templates, admin etc. | * Show data, add data |

* **1. CRUD Operations using Models**
* Let’s break it down like old-school but Gen Z vibes 😎

| **Operation** | **Django Code (Models Way)** | **Meaning** |
| --- | --- | --- |
| **C**reate | Announcement.objects.create(title="New", content="Wow") | Add new row |
| **R**ead (Fetch) | Announcement.objects.all() | Get all rows |
| **U**pdate | a = Announcement.objects.get(id=1); a.title = "Updated"; a.save() | Edit and save |
| **D**elete | a = Announcement.objects.get(id=1); a.delete() | Remove row |

* **📊 2. Real-World Project Model Structure**
* **Example: Blog Project**
* python
* CopyEdit
* class Author(models.Model):
* name = models.CharField(max\_length=100)
* bio = models.TextField()
* class Post(models.Model):
* title = models.CharField(max\_length=200)
* content = models.TextField()
* author = models.ForeignKey(Author, on\_delete=models.CASCADE)
* date\_posted = models.DateTimeField(auto\_now\_add=True)

| **Model** | **Table** | **Purpose** |
| --- | --- | --- |
| Author | author | Stores authors |
| Post | post | Stores blog posts |

* 🎯 See that **ForeignKey**? That’s how we connect tables — old school SQL meets Django magic! 🪄
* **🚀 3. Django QuerySet Cheat Sheet (Fetching Data)**

| **Fetch Type** | **Code Example** |
| --- | --- |
| All records | Announcement.objects.all() |
| One record | Announcement.objects.get(id=1) |
| Filter records | Announcement.objects.filter(title="Sale") |
| Exclude records | Announcement.objects.exclude(title="Old") |
| Order by date | Announcement.objects.order\_by('-date\_posted') |

* ⚠️ **Pro tip:** .get() gives **one** object, .filter() gives a **list** (even if empty!).
* **💪 4. Update & Delete like a Pro**
* **🔄 Update:**
* python
* CopyEdit
* a = Announcement.objects.get(id=1)
* a.title = "Mega Sale"
* a.save()
* **🗑️ Delete:**
* python
* CopyEdit
* a = Announcement.objects.get(id=1)
* a.delete()