Here’s a step-by-step guide to create a Django model with fields:-

- id
- name
- city
- phone
- email

Step 1: Install Django

If Django is not installed:

```
pip install django
```

Step 2: Create Django Project

```
django-admin startproject myproject
```

Move into the project folder:

```
cd myproject
```

Step 3: Create Django App

```
python manage.py startapp users
```

Step 4: Add App to `settings.py`

Open:

`myproject/settings.py`

Add `'users'` inside `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'users',
]
```

Step 5: Create Model

Open:

`users/models.py`

Write this code:

```
from django.db import models

class UserRecord(models.Model):
    name = models.CharField(max_length=100)
    city = models.CharField(max_length=100)
    phone = models.CharField(max_length=15)
    email = models.EmailField(unique=True)

    def __str__(self):
        return self.name
```

Important:

- Django automatically creates an `id` field as primary key.
- You do not need to manually create it unless required.

Step 6: Create Migration

Run:

```
python manage.py makemigrations
```

Example output:

```
D:\>env\scripts\activate

(env) D:\>cd django-warmup

(env) D:\django-warmup>django-admin startproject modelproject

(env) D:\django-warmup>cd modelproject

(env) D:\django-warmup\modelproject>python manage.py startapp users

(env) D:\django-warmup\modelproject>python manage.py makemigrations
Migrations for 'users':
  users\migrations\0001_initial.py
    + Create model UserRecord
```

Step 7: Apply Migration

Run:

```
python manage.py migrate
```

This creates the database table.

```
(env) D:\django-warmup\modelproject>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, users
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
  Applying users.0001_initial... OK
```

Step 8: Register Model in Admin Panel

Open:

`users/admin.py`

Add:

```
from django.contrib import admin
from .models import UserRecord

admin.site.register(UserRecord)
```

Step 9: Create Superuser

Run:

```
python manage.py createsuperuser
```

Enter:

- username
- email
- password

Step 10: Run Server

```
python manage.py runserver
```

Open browser: `http://127.0.0.1:8000/admin`

Django ORM queries using your model:

```
class UserRecord(models.Model):
    name = models.CharField(max_length=100)
    city = models.CharField(max_length=100)
    phone = models.CharField(max_length=15)
    email = models.EmailField(unique=True)
```

1. How to Open Django Shell

□ Step 1: Go to your project folder

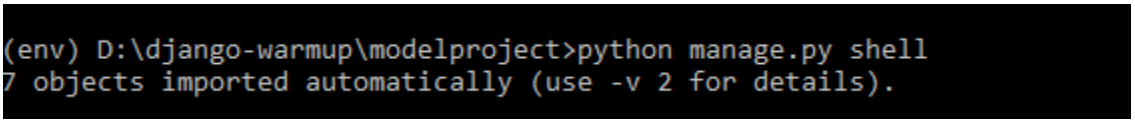
Make sure you are inside the folder where `manage.py` is located.

Example:

```
cd myproject
```

□ Step 2: Run Django shell command

```
python manage.py shell
```



Import model first:

```
from users.models import UserRecord
```

```
>>> from users.models import UserRecord
>>>
```

In this import:

```
from users.models import UserRecord
```

□ What is `users` here?

□ `users` is the **Django app name**.

In Django, a project is made of multiple **apps**, and each app has its own `models.py`, `views.py`, etc.

So this means:

- `users` → app folder (like `users/`)
- `models` → file inside that app (`models.py`)
- `UserRecord` → a model (database table) inside `models.py`

1. `all()`

Used to fetch ALL records from the table.

Syntax

```
UserRecord.objects.all()
```

Example

```
users = UserRecord.objects.all()

for user in users:
    print(user.name)
```

As shown

```
>>> records = UserRecord.objects.all()
>>>
>>> for r in records:
...     print(r.id, r.name, r.city)
...
1 John Mumbai
2 Amit Delhi
3 Rahul Pune
4 Sneha Mumbai
>>> UserRecord.objects.get(id=1)
<UserRecord: John>
>>> UserRecord.objects.get(email="john@gmail.com")
<UserRecord: John>
```

SQL Equivalent

```
SELECT * FROM users_userrecord;
```

Returns

- QuerySet (collection of objects)

Use Case

- Show all users
- Display table data

2. `filter()`

Used to fetch records matching conditions.

Syntax

```
UserRecord.objects.filter(field=value)
```

Example

```
UserRecord.objects.filter(city="Mumbai")
```

Multiple Conditions

```
UserRecord.objects.filter(city="Mumbai", name="John")
```

SQL Equivalent

```
SELECT * FROM users_userrecord
WHERE city='Mumbai';
```

Returns

- QuerySet

Important

Even if only one record matches, `filter()` still returns a QuerySet.

---

3. `get()`

Used to fetch ONE exact record.

Syntax

```
UserRecord.objects.get(id=1)
```

Example

```
user = UserRecord.objects.get(email="john@gmail.com")

print(user.name)
```

SQL Equivalent

```
SELECT * FROM users_userrecord
WHERE email='john@gmail.com';
```

Returns

- Single object

Errors

If no record found:

```
UserRecord.DoesNotExist
```

If multiple records found:

```
MultipleObjectsReturned
```

Best Use

- Primary key lookup
- Unique field lookup

---

4. `create()`

Used to insert a new record.

Syntax

```
UserRecord.objects.create(...)
```

Example

```
UserRecord.objects.create(
    name="Rahul",
    city="Pune",
    phone="9999999999",
    email="rahul@gmail.com"
)
```

```
>>> UserRecord.objects.create(
...     name="John",
...     city="Mumbai",
...     phone="9876543210",
...     email="john@gmail.com"
... )
```

SQL Equivalent

```
INSERT INTO users_userrecord (...)
VALUES (...);
```

Returns

- Created object

---

5. `update()`

Used to update existing records.

Syntax

```
UserRecord.objects.filter(...).update(...)
```

Example

```
UserRecord.objects.filter(id=1).update(city="Delhi")
```

SQL Equivalent

```
UPDATE users_userrecord
SET city='Delhi'
WHERE id=1;
```

Returns

- Number of updated rows

Important

`update()` does NOT call `save()` method.

---

6. `delete()`

Used to delete records.

Syntax

```
object.delete()
```

OR

```
QuerySet.delete()
```

Example 1: Delete Single Record

```
user = UserRecord.objects.get(id=1)
user.delete()
```

Example 2: Delete Multiple Records

```
UserRecord.objects.filter(city="Delhi").delete()
```

SQL Equivalent

```
DELETE FROM users_userrecord
WHERE city='Delhi';
```

---

7. `exclude()`

Opposite of `filter()`.

Returns records NOT matching condition.

Syntax

```
UserRecord.objects.exclude(city="Mumbai")
```

SQL Equivalent

```
SELECT * FROM users_userrecord
WHERE NOT city='Mumbai';
```

Returns

- QuerySet

---

8. `count()`

Counts records.

Syntax

```
UserRecord.objects.count()
```

Example

```
UserRecord.objects.filter(city="Mumbai").count()
```

SQL Equivalent

```
SELECT COUNT(*) FROM users_userrecord;
```

Returns

- Integer

---

9. `exists()`

Checks whether records exist.

Syntax

```
UserRecord.objects.filter(...).exists()
```

Example

```
UserRecord.objects.filter(email="john@gmail.com").exists()
```

Returns

- `True`
- `False`

SQL Equivalent

```
SELECT EXISTS(...);
```

Best Use

Efficient existence checking.

---

10. `order_by()`

Sorts records.

Ascending

```
UserRecord.objects.order_by("name")
```

Descending

```
UserRecord.objects.order_by("-id")
```

SQL Equivalent

```
SELECT * FROM users_userrecord
ORDER BY name ASC;
```

---

11. `values()`

Returns dictionary data instead of objects.

Syntax

```
UserRecord.objects.values("name", "city")
```

Example Output

```
[
    {'name': 'John', 'city': 'Mumbai'},
    {'name': 'Rahul', 'city': 'Pune'}
]
```

Best Use

- APIs
- JSON responses
- Optimized queries

---

12. `annotate()`

Used for aggregation/grouping.

Usually used with `Count`, `Sum`, `Avg`.

Import

```
from django.db.models import Count
```

Example

Count users per city:

```
UserRecord.objects.values("city").annotate(total=Count("id"))
```

Output

```
[
    {'city': 'Mumbai', 'total': 5},
    {'city': 'Pune', 'total': 2}
]
```

SQL Equivalent

```
SELECT city, COUNT(id)
FROM users_userrecord
GROUP BY city;
```

---

13. `Q()`

Used for complex queries (OR, AND, NOT).

Import

```
from django.db.models import Q
```

OR Query

```
UserRecord.objects.filter(
    Q(city="Mumbai") | Q(city="Delhi")
)
```

SQL Equivalent

```
WHERE city='Mumbai' OR city='Delhi'
```

AND Query

```
UserRecord.objects.filter(
    Q(city="Mumbai") & Q(name="John")
)
```

NOT Query

```
UserRecord.objects.filter(
    ~Q(city="Mumbai")
)
```

---

14. `get_or_create()`

Gets object if exists, otherwise creates it.

Syntax

```
Model.objects.get_or_create(
    field=value,
    defaults={}
)
```

Example

```
user, created = UserRecord.objects.get_or_create(
    email="test@gmail.com",
    defaults={
        "name": "Test",
        "city": "Mumbai",
        "phone": "8888888888"
    }
)
```

Returns

```
(object, created)
```

Where:

- `object` = model object
- `created` = True/False

If Record Exists

```
created = False
```

If Record Created

```
created = True
```

Important Difference Table

Method	Returns	Multiple Records	Error Possible
`all()`	QuerySet	Yes	No
`filter()`	QuerySet	Yes	No
`get()`	Single Object	No	Yes
`create()`	Object	No	Possible
`update()`	Count	Yes	No
`delete()`	Tuple	Yes	No
`exclude()`	QuerySet	Yes	No
`count()`	Integer	Yes	No
`exists()`	Boolean	Yes	No
`order_by()`	QuerySet	Yes	No
`values()`	Dictionary QuerySet	Yes	No
`annotate()`	Grouped QuerySet	Yes	No
`Q()`	Complex Query	Yes	No
`get_or_create()`	Tuple	No	Possible