Models relationship

1. **Create Virtual Environment**
   - python -m venv env
2. **Activate Virtual Environment**
   - env\Scripts\activat
3. **Install Django**
   - pip install Django
4. **Create Django Project**
   - django-admin startproject project
   - cd project
5. **Create App**
   - python manage.py startapp studentapp
6. Add app in **settings.py**

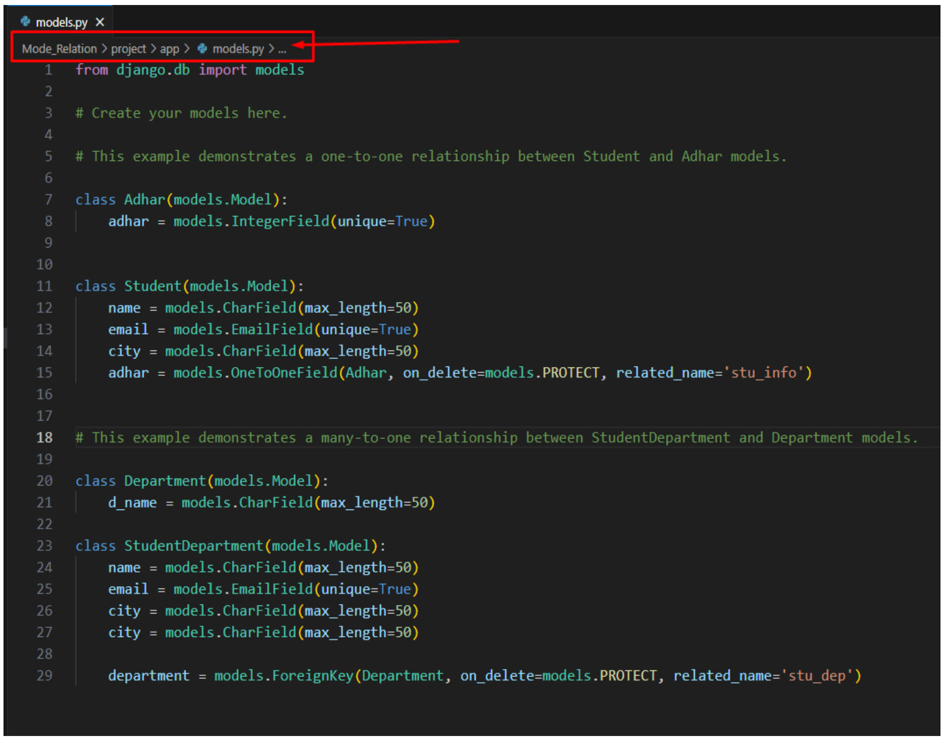
```
# project/settings.py
INSTALLED_APPS = [
    ...
    …
    'app',
]
```

7. **Source Code of app/models.py (Copied Text for Easy Reference)**
   Below is the same code shown in the screenshot.
   You can copy and paste it directly in your models.py file.

   Code >>>>>>>

```
from django.db import models

# Create your models here.

# This example demonstrates a one-to-one relationship between Student and Adhar models.

class Adhar(models.Model):
    adhar = models.IntegerField(unique=True)

class Student(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
    city = models.CharField(max_length=50)
    adhar = models.OneToOneField(Adhar, on_delete=models.PROTECT, related_name='stu_info')

# This example demonstrates a many-to-one relationship between StudentDepartment and Department
models.

class Department(models.Model):
    d_name = models.CharField(max_length=50)

class StudentDepartment(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
```

```
city = models.CharField(max_length=50)
city = models.CharField(max_length=50)

department = models.ForeignKey(Department, on_delete=models.PROTECT, related_name='stu_dep')
```



models.py ×

Mode_Relation > project > app > models.py > ...

```python
from django.db import models

# Create your models here.

# This example demonstrates a one-to-one relationship between Student and Adhar models.

class Adhar(models.Model):
    adhar = models.IntegerField(unique=True)


class Student(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
    city = models.CharField(max_length=50)
    adhar = models.OneToOneField(Adhar, on_delete=models.PROTECT, related_name='stu_info')


# This example demonstrates a many-to-one relationship between StudentDepartment and Department models.

class Department(models.Model):
    d_name = models.CharField(max_length=50)

class StudentDepartment(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
    city = models.CharField(max_length=50)
    city = models.CharField(max_length=50)

    department = models.ForeignKey(Department, on_delete=models.PROTECT, related_name='stu_dep')
```

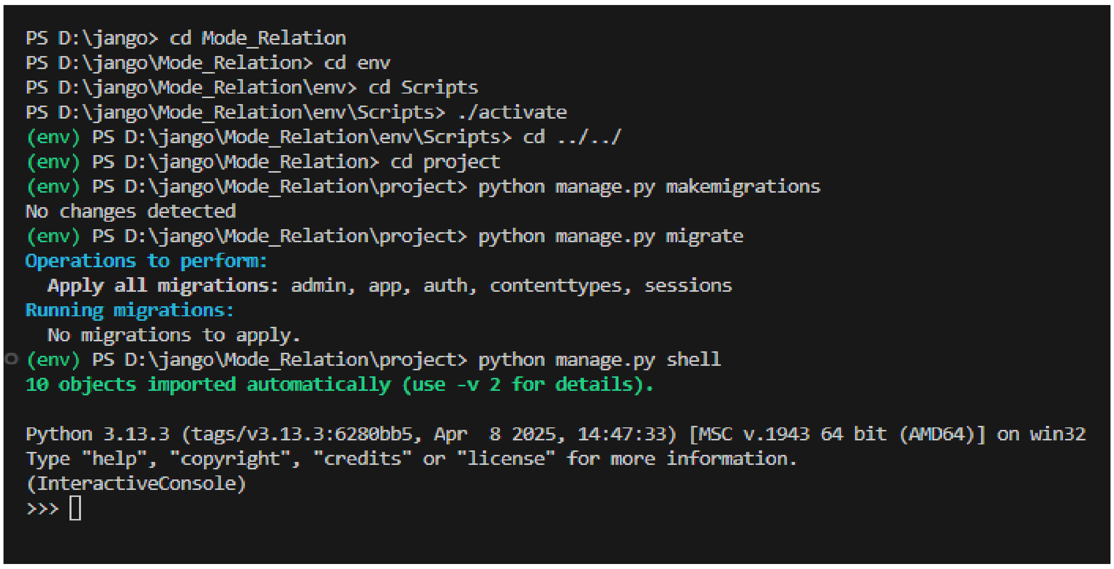
8. Database Migration Commands (After Writing models.py)

- ➢ python manage.py makemigrations
- ➢ python manage.py migrate

Next Steps: Insert Data into Tables Using Django Shell

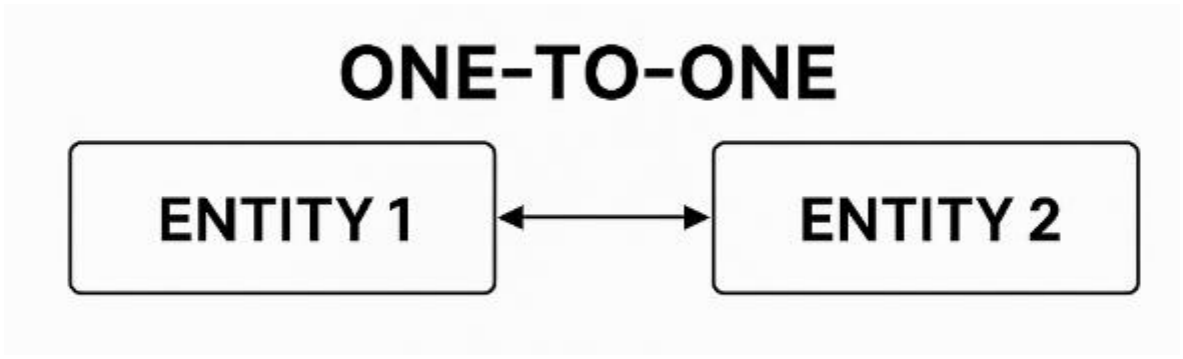
Shell open karne ke liye command:

- ➢ python manage.py shell
  commands till shell in below image



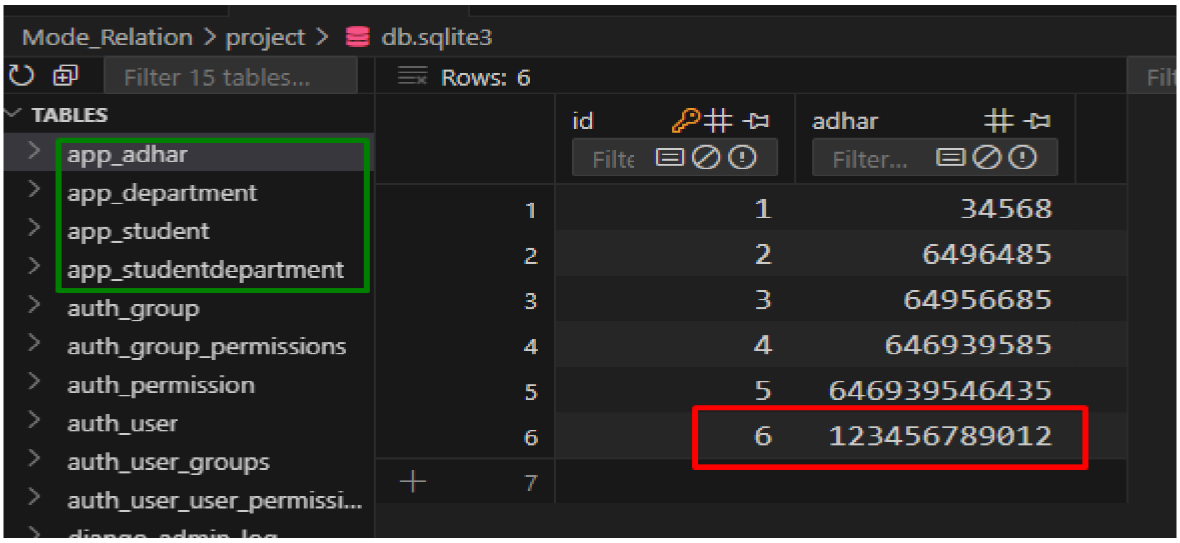
```
PS D:\jango> cd Mode_Relation
PS D:\jango\Mode_Relation> cd env
PS D:\jango\Mode_Relation\env> cd Scripts
PS D:\jango\Mode_Relation\env\Scripts> ./activate
(env) PS D:\jango\Mode_Relation\env\Scripts> cd ../../
(env) PS D:\jango\Mode_Relation> cd project
(env) PS D:\jango\Mode_Relation\project> python manage.py makemigrations
No changes detected
(env) PS D:\jango\Mode_Relation\project> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, app, auth, contenttypes, sessions
Running migrations:
  No migrations to apply.
(env) PS D:\jango\Mode_Relation\project> python manage.py shell
10 objects imported automatically (use -v 2 for details).

Python 3.13.3 (tags/v3.13.3:6280bb5, Apr  8 2025, 14:47:33) [MSC v.1943 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> 
```

1. One-to-One Relationship: Student & Adhar



- from app.models import Student, Adhar

  # Adhar object create karo
- a1 = Adhar.objects.create(adhar=123456789012)


Mode_Relation > project > db.sqlite3

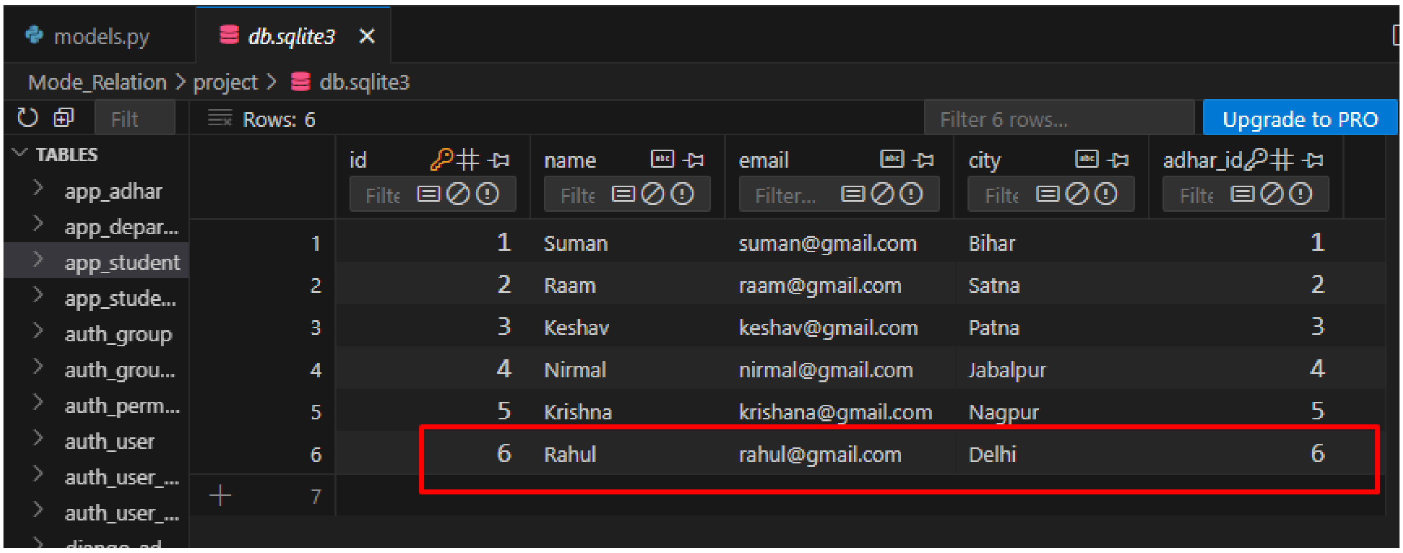
Filter 15 tables...

Rows: 6

TABLES
- app_adhar
- app_department
- app_student
- app_studentdepartment
- auth_group
- auth_group_permissions
- auth_permission
- auth_user
- auth_user_groups
- auth_user_user_permissi...

	id	adhar
1	1	34568
2	2	6496485
3	3	64956685
4	4	646939585
5	5	646939546435
6	6	123456789012
7		

  # Student object create karo aur Adhar assign karo
- s1 = Student.objects.create(name="Rahul", email="rahul@gmail.com", city="Delhi", adhar=a1)


models.py | db.sqlite3

Mode_Relation > project > db.sqlite3

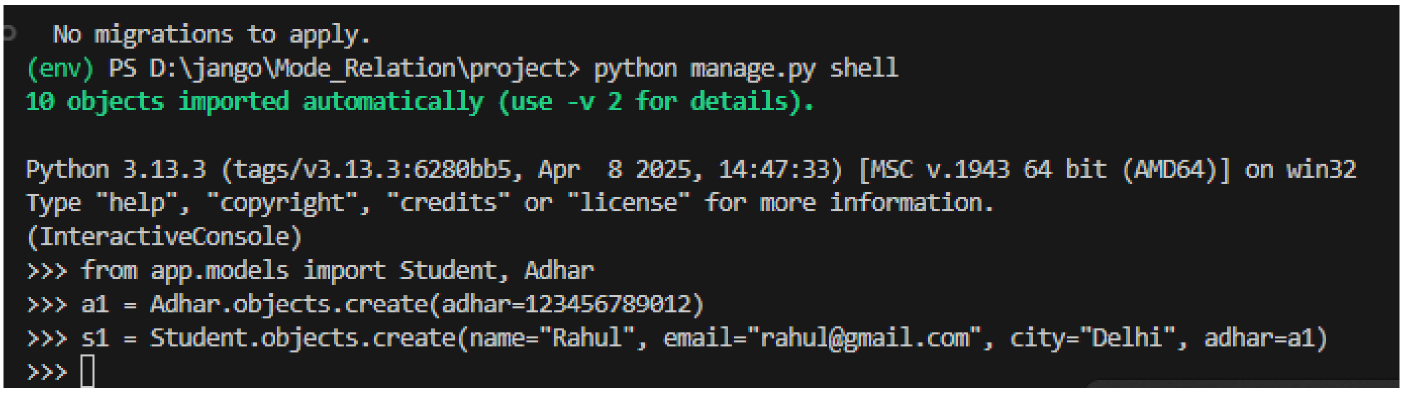
Rows: 6

Filter 6 rows...

Upgrade to PRO

TABLES
- app_adhar
- app_depar...
- app_student
- app_stude...
- auth_group
- auth_grou...
- auth_perm...
- auth_user
- auth_user_...
- auth_user_...

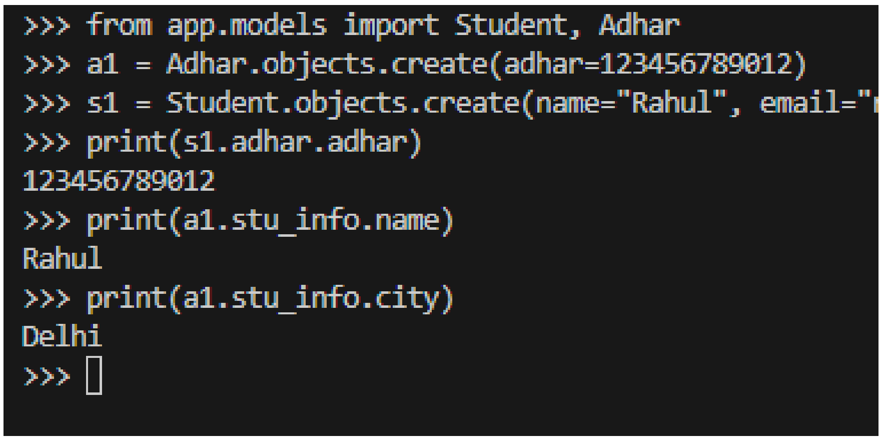
	id	name	email	city	adhar_id
1	1	Suman	suman@gmail.com	Bihar	1
2	2	Raam	raam@gmail.com	Satna	2
3	3	Keshav	keshav@gmail.com	Patna	3
4	4	Nirmal	nirmal@gmail.com	Jabalpur	4
5	5	Krishna	krishana@gmail.com	Nagpur	5
6	6	Rahul	rahul@gmail.com	Delhi	6
7					



```
No migrations to apply.
(env) PS D:\jango\Mode_Relation\project> python manage.py shell
10 objects imported automatically (use -v 2 for details).

Python 3.13.3 (tags/v3.13.3:6280bb5, Apr  8 2025, 14:47:33) [MSC v.1943 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from app.models import Student, Adhar
>>> a1 = Adhar.objects.create(adhar=123456789012)
>>> s1 = Student.objects.create(name="Rahul", email="rahul@gmail.com", city="Delhi", adhar=a1)
>>>
```

# Access One-to-One data

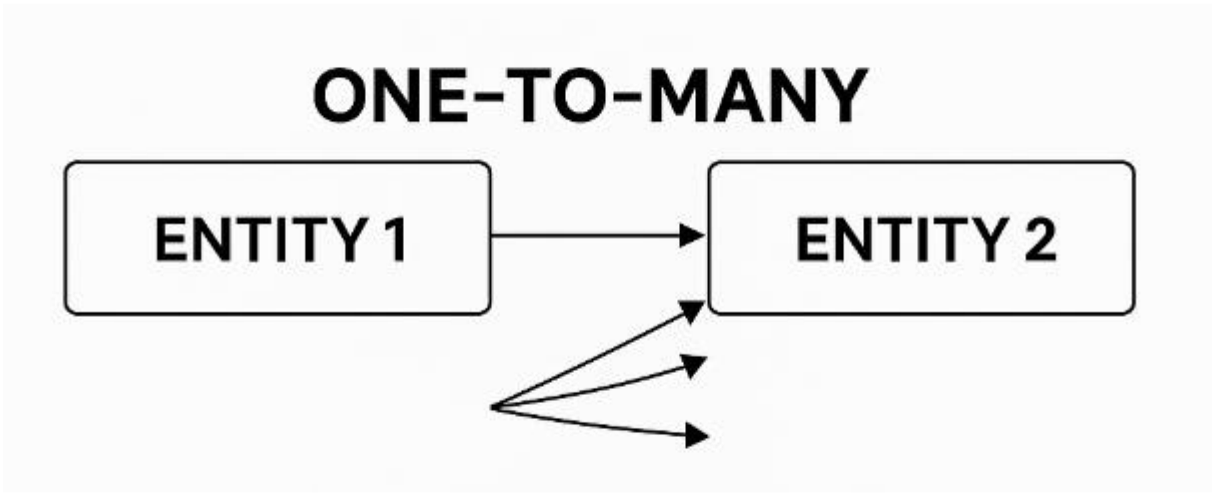
- **print(s1.adhar.adhar)**
- **print(a1.stu_info.name) # using related_name**



```
>>> from app.models import Student, Adhar
>>> a1 = Adhar.objects.create(adhar=123456789012)
>>> s1 = Student.objects.create(name="Rahul", email="r
>>> print(s1.adhar.adhar)
123456789012
>>> print(a1.stu_info.name)
Rahul
>>> print(a1.stu_info.city)
Delhi
>>>
```

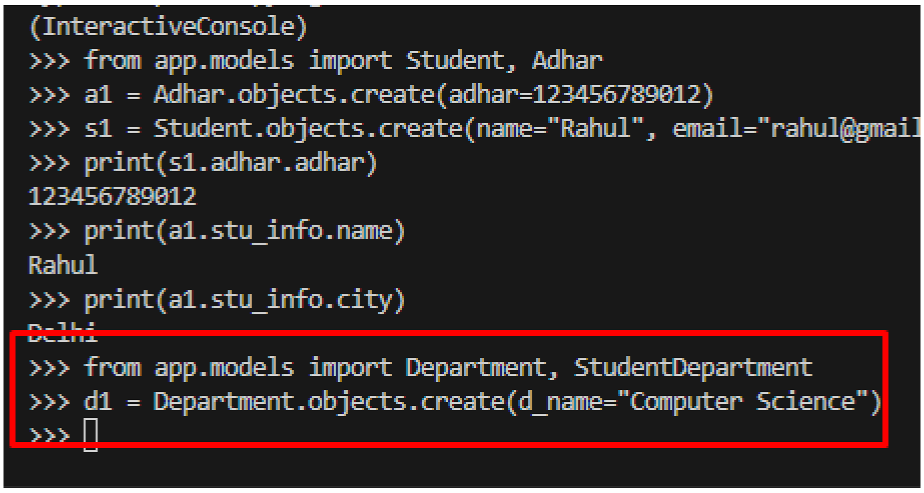
---

2. <u>Many-to-One Relationship: StudentDepartment & Department</u>



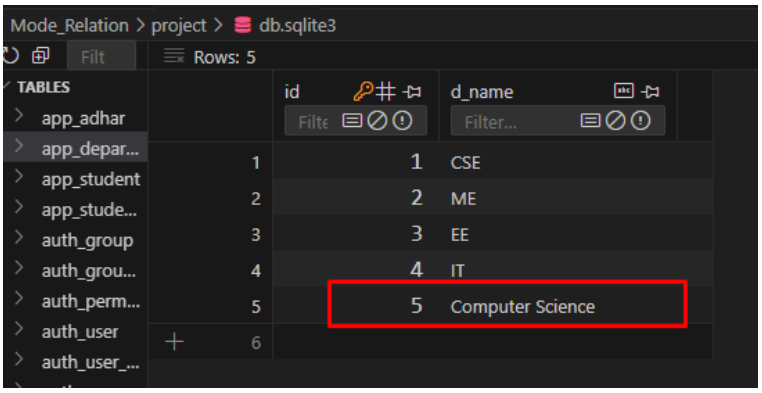
- from app.models import Department, StudentDepartment

# Department create karo
- d1 = Department.objects.create(d_name="Computer Science")



```
(InteractiveConsole)
>>> from app.models import Student, Adhar
>>> a1 = Adhar.objects.create(adhar=123456789012)
>>> s1 = Student.objects.create(name="Rahul", email="rahul@gmail
>>> print(s1.adhar.adhar)
123456789012
>>> print(a1.stu_info.name)
Rahul
>>> print(a1.stu_info.city)
Delhi
>>> from app.models import Department, StudentDepartment
>>> d1 = Department.objects.create(d_name="Computer Science")
>>>
```

Refresh the data and check it



Mode_Relation > project > db.sqlite3

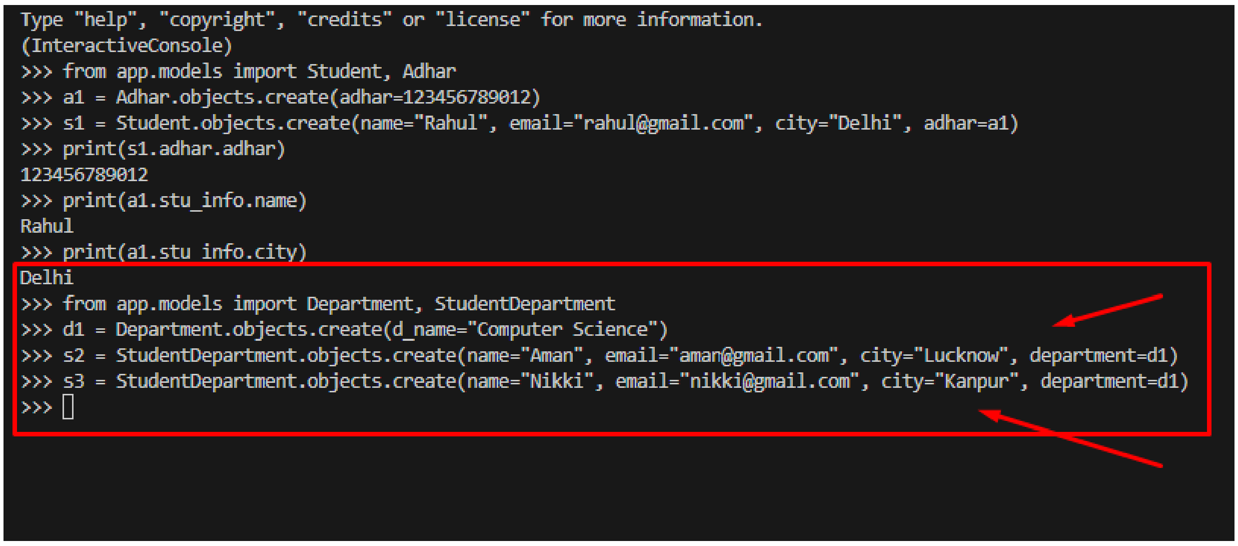
Rows: 5

TABLES
- app_adhar
- app_depar...
- app_student
- app_stude...
- auth_group
- auth_grou...
- auth_perm...
- auth_user
- auth_user_...

	id	d_name
1	1	CSE
2	2	ME
3	3	EE
4	4	IT
5	5	Computer Science
6		

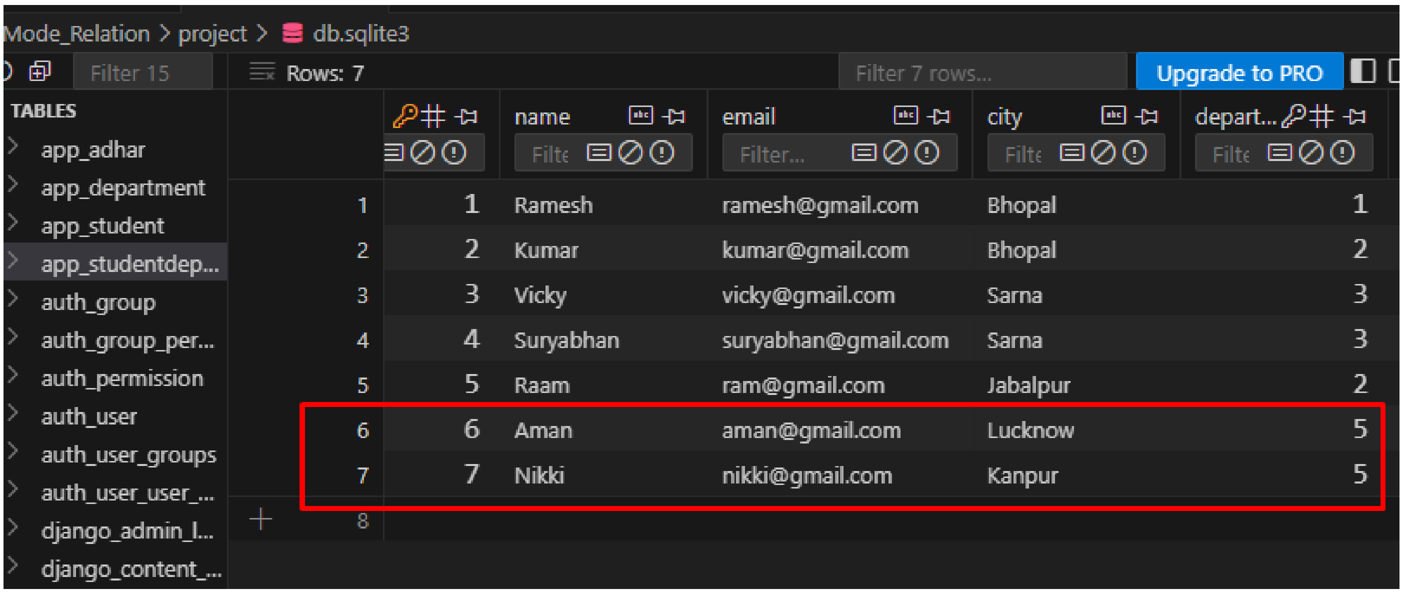
# StudentDepartment create karo aur department assign karo

- s2 = StudentDepartment.objects.create(name="Aman", email="aman@gmail.com", city="Lucknow", department=d1)
- s3 = StudentDepartment.objects.create(name="Nikki", email="nikki@gmail.com", city="Kanpur", department=d1)



```
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from app.models import Student, Adhar
>>> a1 = Adhar.objects.create(adhar=123456789012)
>>> s1 = Student.objects.create(name="Rahul", email="rahul@gmail.com", city="Delhi", adhar=a1)
>>> print(s1.adhar.adhar)
123456789012
>>> print(a1.stu_info.name)
Rahul
>>> print(a1.stu_info.city)
Delhi
>>> from app.models import Department, StudentDepartment
>>> d1 = Department.objects.create(d_name="Computer Science")
>>> s2 = StudentDepartment.objects.create(name="Aman", email="aman@gmail.com", city="Lucknow", department=d1)
>>> s3 = StudentDepartment.objects.create(name="Nikki", email="nikki@gmail.com", city="Kanpur", department=d1)
>>>
```

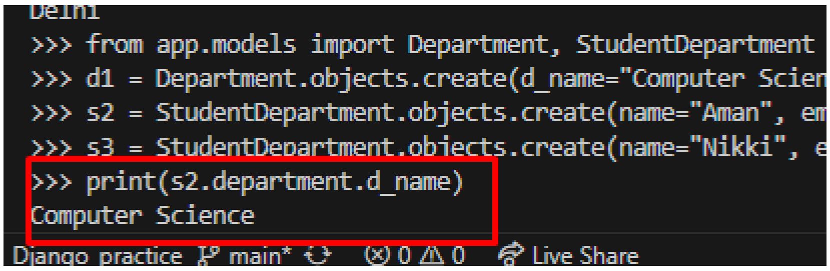
Refresh the studentdepartment data and check it



	#	name	email	city	depart...
1	1	Ramesh	ramesh@gmail.com	Bhopal	1
2	2	Kumar	kumar@gmail.com	Bhopal	2
3	3	Vicky	vicky@gmail.com	Sarna	3
4	4	Suryabhan	suryabhan@gmail.com	Sarna	3
5	5	Raam	ram@gmail.com	Jabalpur	2
6	6	Aman	aman@gmail.com	Lucknow	5
7	7	Nikki	nikki@gmail.com	Kanpur	5
8					

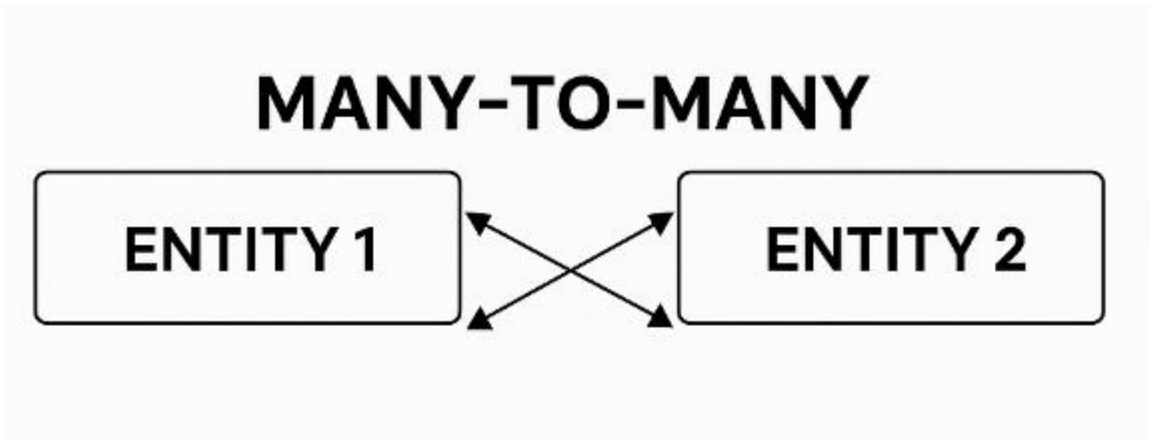
# Access Many-to-One data

➢ print(s2.department.d_name)



```
Delhi
>>> from app.models import Department, StudentDepartment
>>> d1 = Department.objects.create(d_name="Computer Scien
>>> s2 = StudentDepartment.objects.create(name="Aman", em
>>> s3 = StudentDepartment.objects.create(name="Nikki", e
>>> print(s2.department.d_name)
Computer Science
```

3. **Many-to-Many relationship between Vehicle and Fuel**



Models Involved:

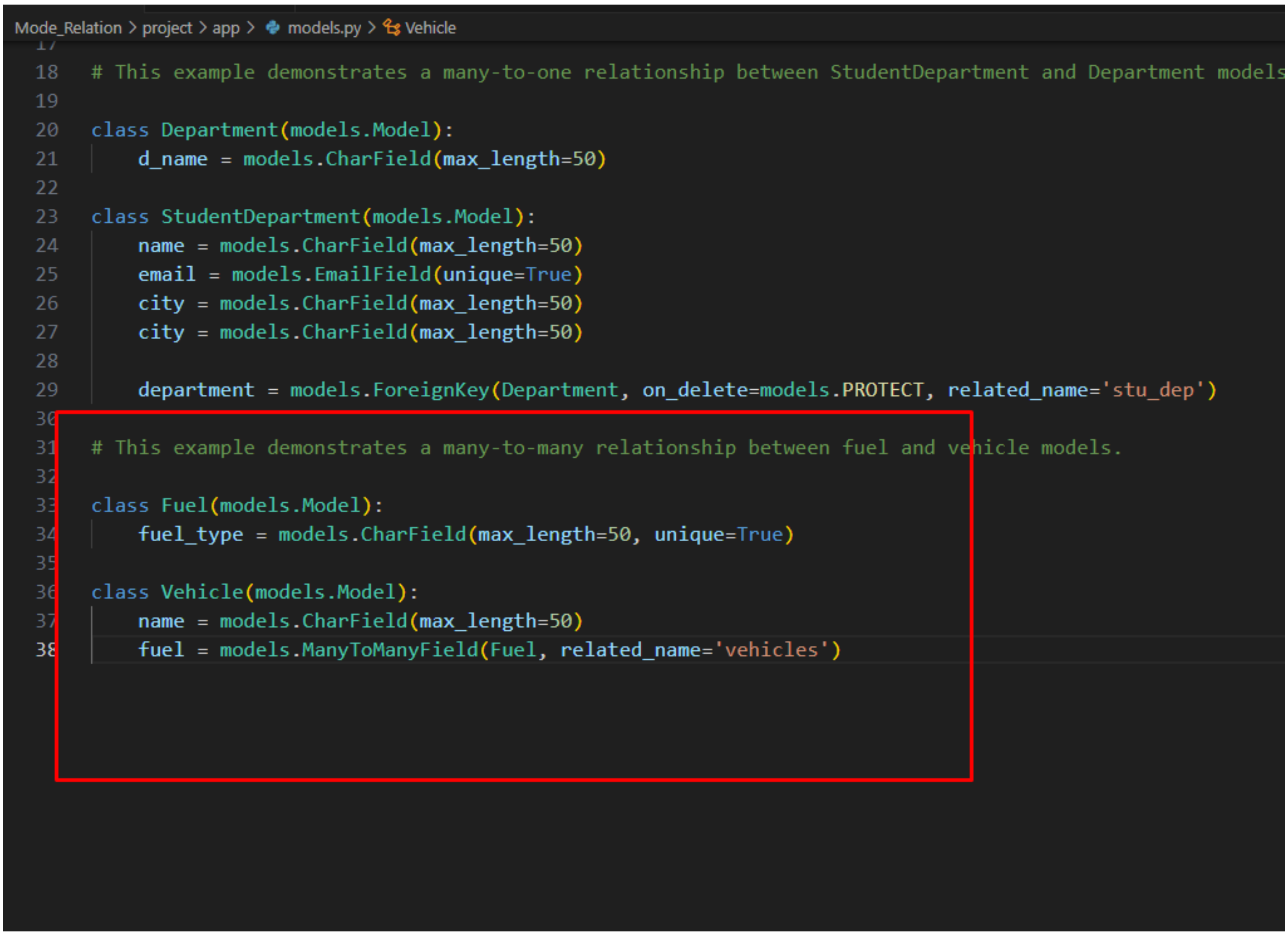
- **Fuel:** Represents types of fuel (Petrol, Diesel, etc.)
- **Vehicle:** Represents vehicles that can run on one or more types of fuels.

Use-case Example:

- Tata Nexon can run on Petrol and CNG.
- Diesel can be used by multiple vehicles like Scorpio, Thar, etc.

Many-to-Many relationships are ideal for modeling complex real-world scenarios where multiple entries from both sides can relate to each other.

Add the bellow code in previous models.py file.



```
# This example demonstrates a many-to-one relationship between StudentDepartment and Department models

class Department(models.Model):
    d_name = models.CharField(max_length=50)

class StudentDepartment(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
    city = models.CharField(max_length=50)
    city = models.CharField(max_length=50)

    department = models.ForeignKey(Department, on_delete=models.PROTECT, related_name='stu_dep')

# This example demonstrates a many-to-many relationship between fuel and vehicle models.

class Fuel(models.Model):
    fuel_type = models.CharField(max_length=50, unique=True)

class Vehicle(models.Model):
    name = models.CharField(max_length=50)
    fuel = models.ManyToManyField(Fuel, related_name='vehicles')
```

Below is the same code as shown in the screenshot.
You can copy and paste it directly into your models.py file

# This example demonstrates a many-to-many relationship between fuel and vehicle models.

class Fuel(models.Model):

fuel_type = models.CharField(max_length=50, unique=True)

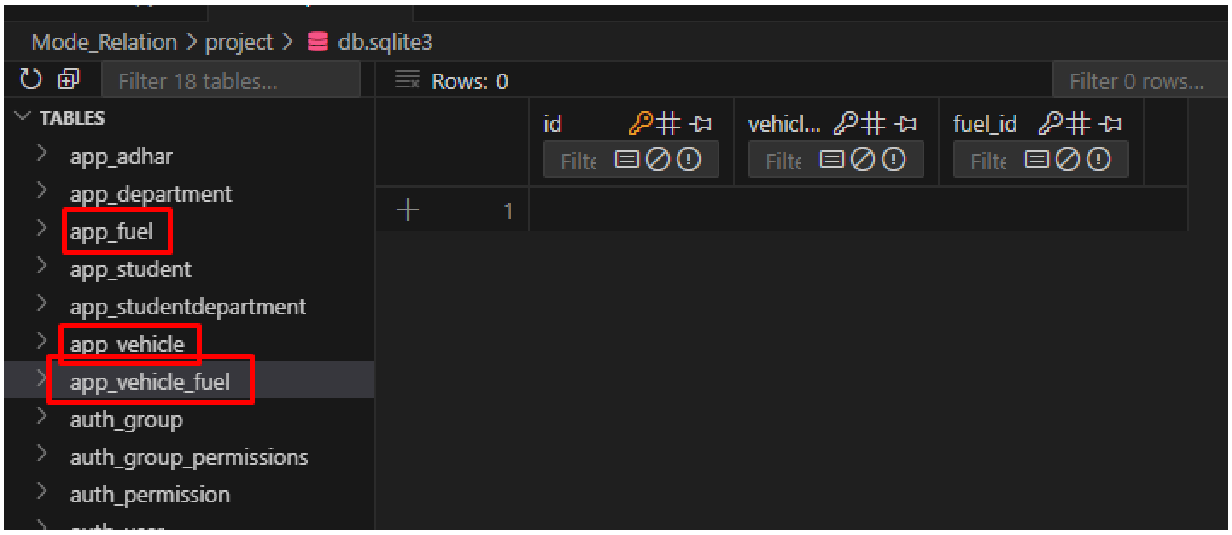
class Vehicle(models.Model):

```
name = models.CharField(max_length=50)

fuel = models.ManyToManyField(Fuel, related_name='vehicles')
```

- Make Migrations
  - python manage.py makemigrations
  - python manage.py migrate

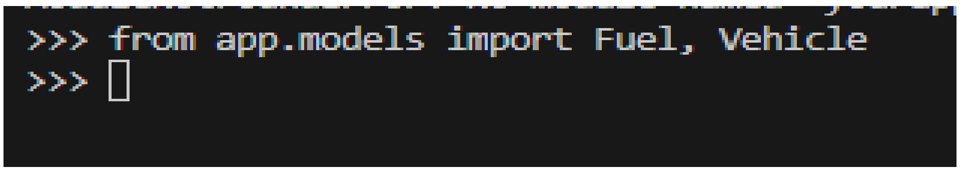
After running the commands:

Django automatically creates the necessary tables in the database based on the models and relationships defined. For each type of relationship, different numbers and types of tables are created:



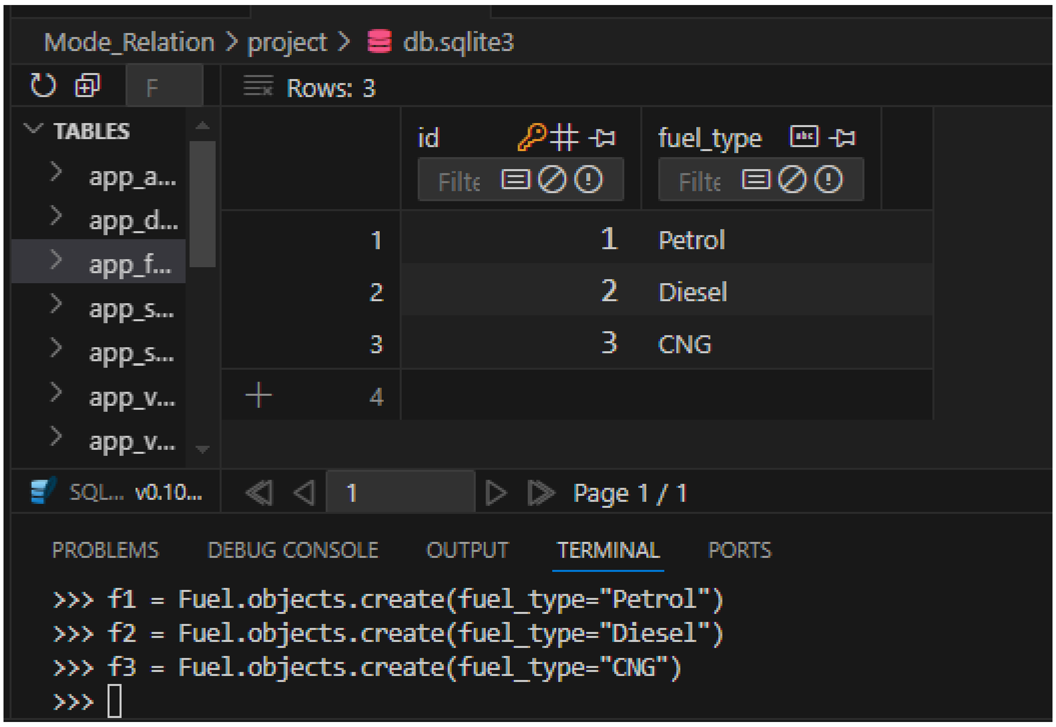
- Open Django Shell
  - python manage.py shell

- Import the models:
  - from app.models import Fuel, Vehicle



```
>>> from app.models import Fuel, Vehicle
>>> 
```

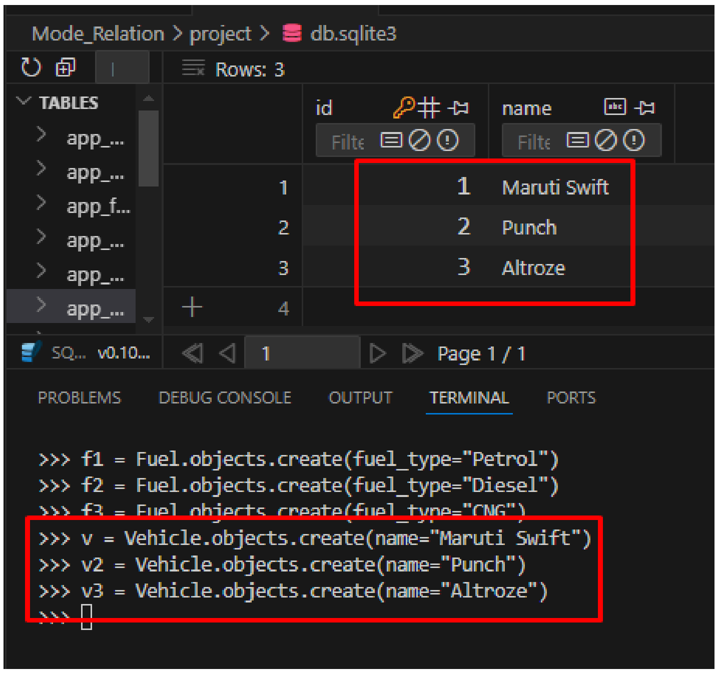
- **Create Fuel objects:**
  - **f1 = Fuel.objects.create(fuel_type="Petrol")**
  - **f2 = Fuel.objects.create(fuel_type="Diesel")**
  - **f3 = Fuel.objects.create(fuel_type="CNG")**



	id	fuel_type
1	1	Petrol
2	2	Diesel
3	3	CNG
4		

```
>>> f1 = Fuel.objects.create(fuel_type="Petrol")
>>> f2 = Fuel.objects.create(fuel_type="Diesel")
>>> f3 = Fuel.objects.create(fuel_type="CNG")
>>> 
```

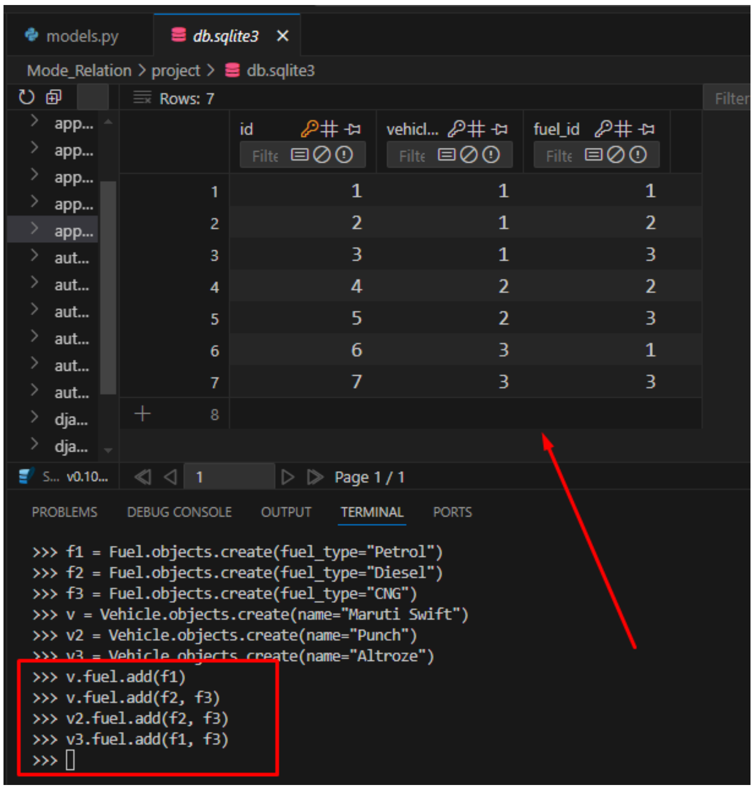
- **Create a Vehicle object:**
  - **v = Vehicle.objects.create(name="Maruti Swift")**
  - **v2 = Vehicle.objects.create(name="Punch")**
  - **v3 = Vehicle.objects.create(name="Altroze")**



	id	name
1	1	Maruti Swift
2	2	Punch
3	3	Altroze
4		

```
>>> f1 = Fuel.objects.create(fuel_type="Petrol")
>>> f2 = Fuel.objects.create(fuel_type="Diesel")
>>> f3 = Fuel.objects.create(fuel_type="CNG")
>>> v = Vehicle.objects.create(name="Maruti Swift")
>>> v2 = Vehicle.objects.create(name="Punch")
>>> v3 = Vehicle.objects.create(name="Altroze")
>>> 
```

- **Assign fuels to vehicle:**
  - **v.fuel.add(f1)** **# Add one fuel type**
  - **v.fuel.add(f2, f3)** **# Add multiple fuel types**
  - **v2.fuel.add(f2, f3)**
  - **v3.fuel.add(f1, f3)**

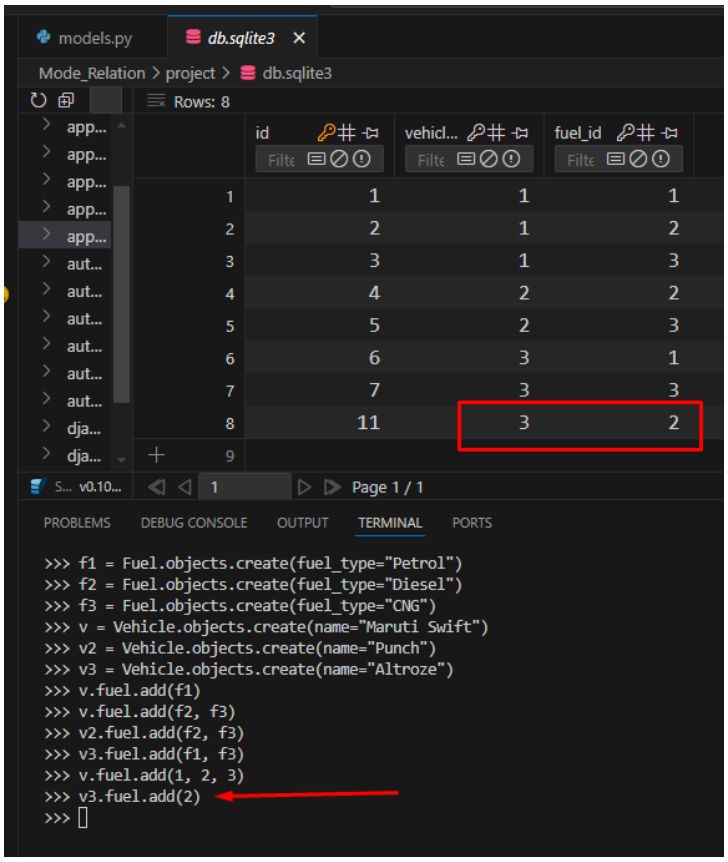


	id	vehicl...	fuel_id
1	1	1	1
2	2	1	2
3	3	1	3
4	4	2	2
5	5	2	3
6	6	3	1
7	7	3	3

```
>>> f1 = Fuel.objects.create(fuel_type="Petrol")
>>> f2 = Fuel.objects.create(fuel_type="Diesel")
>>> f3 = Fuel.objects.create(fuel_type="CNG")
>>> v = Vehicle.objects.create(name="Maruti Swift")
>>> v2 = Vehicle.objects.create(name="Punch")
>>> v3 = Vehicle.objects.create(name="Altroze")
>>> v.fuel.add(f1)
>>> v.fuel.add(f2, f3)
>>> v2.fuel.add(f2, f3)
>>> v3.fuel.add(f1, f3)
>>> 
```

Or by IDs:

- **v3.fuel.add(2)**



	id	vehicl...	fuel_id
1	1	1	1
2	2	1	2
3	3	1	3
4	4	2	2
5	5	2	3
6	6	3	1
7	7	3	3
8	11	3	2

```
>>> f1 = Fuel.objects.create(fuel_type="Petrol")
>>> f2 = Fuel.objects.create(fuel_type="Diesel")
>>> f3 = Fuel.objects.create(fuel_type="CNG")
>>> v = Vehicle.objects.create(name="Maruti Swift")
>>> v2 = Vehicle.objects.create(name="Punch")
>>> v3 = Vehicle.objects.create(name="Altroze")
>>> v.fuel.add(f1)
>>> v.fuel.add(f2, f3)
>>> v2.fuel.add(f2, f3)
>>> v3.fuel.add(f1, f3)
>>> v.fuel.add(1, 2, 3)
>>> v3.fuel.add(2)
>>> 
```

- Querying Many-to-Many
    - All fuels of a vehicle:

      ```
      >>>v.fuel.all()
      ```
    - Remove or Clear

      ```
      >>> v.fuel.remove(f1)
      >>> v.fuel.clear()
      ```