**Example 1**

**Terminal**

python manage.py migrate

python manage.py createsuperuser

**models.py**

from django.db import models

class Emp(models.Model):

  firstname = models.CharField(max\_length=255)

  lastname = models.CharField(max\_length=255)

  salary = models.IntegerField()

def \_\_str\_\_(self):

        return self.firstname+ ":"+ self.salary

**Terminal**

python manage.py makemigrations

python manage.py migrate

python manage.py shell

**Insert data using terminal**

>>> from myapp.models import Emp

>>> Emp.objects.all()

<QuerySet []>

>>> Emp.objects.create(firstname='Rinki',lastname='Kumari',salary=30000)

>>> Emp.objects.all()

<QuerySet [<Emp: Emp object (1)>]>

>>>

**admin.py**

from django.contrib import admin

from .models import Emp

admin.site.register(Emp)

Now install the extension SQLite in VS code

You will see db.sqlite3 in the Solution Explorer. Expand it and find the name of your table. In this case, the name is Emp, so you should be able to see something like myapp\_emp. Click on the arrow given at the RHS of the table name. SQLite window will open and show you your table with all data you have added.

After this, open localhost:8000.admin. Here you can perform insert, update and delete operations

**Update data using terminal**

>>> from myapp.models import Emp

>>> e=Emp.objects.get(pk=2)

>>> e.salary=45000

>>> e.save()

>>>

**Delete data using terminal**

>>> e=Emp.objects.get(pk=1)

>>> e.delete()

(1, {'myapp.Emp': 1})

>>>

**Example 1(Foreign key)**

**models.py**

from django.db import models

class Author(models.Model):

    name = models.CharField(max\_length=100)

    email = models.EmailField()

    description = models.TextField()

    def \_\_str\_\_(self):

        return self.name

class Book(models.Model):

    title = models.CharField(max\_length=100)

    # author = models.ForeignKey(Author, on\_delete=models.CASCADE)

    author = models.ForeignKey(Author, on\_delete=models.SET\_NULL, null = True)

    published\_date = models.DateField()

    def \_\_str\_\_(self):

        return self.title

**Terminal**

python manage.py makemigrations

python manage.py migrate

python manage.py shell

**Insert data using terminal**

>>> from myapp.models import Author, Book

>>> author1 = Author.objects.create(name="J.K. Rowling", email="jkrowling@example.com", description="Author of the Harry Potter series")

>>> author2 = Author.objects.create(name="Stephen King", email="stephenking@example.com", description="Renowned horror fiction author")

>>> book1 = Book.objects.create(title="Harry Potter and the Philosopher's Stone", author=author1, published\_date="1997-06-26")

>>> book2 = Book.objects.create(title="The Shining", author=author2, published\_date="1977-01-28")

NOTE:

In Django, the **on\_delete** parameter of a ForeignKey field specifies the behavior to adopt when the referenced object (in this case, an instance of the **Author** model) is deleted.

The **models.CASCADE** option specifies that when the referenced object is deleted, also delete the objects that have a ForeignKey pointing to it. In our example, it means that if an **Author** instance is deleted, all associated **Book** instances (which have a ForeignKey pointing to that author) will also be deleted.

Here's a breakdown of some common options for the **on\_delete** parameter:

* **models.CASCADE**: Deletes the objects that have a ForeignKey pointing to the deleted object.
* **models.PROTECT**: Prevents deletion of the referenced object by raising a **ProtectedError**.
* **models.SET\_NULL**: Sets the ForeignKey to NULL when the referenced object is deleted. Note that you need to ensure the field is nullable.
* **models.SET\_DEFAULT**: Sets the ForeignKey to its default value when the referenced object is deleted.
* **models.SET**: Sets the ForeignKey to the value passed to the **SET()** function.
* **models.DO\_NOTHING**: Takes no action when the referenced object is deleted. This can potentially leave dangling references in the database.

It's important to choose the appropriate **on\_delete** behavior based on the requirements of your application and the integrity constraints of your database schema.