Performing CURD operations by using python shell or Console (by using ORM Queries):

The Django web framework includes a default object-relational mapping layer (ORM) that can be used to interact with data from various relational databases such as SQLite, PostgreSQL, and MySQL.

Django allows us to add, delete, modify and query objects, using an API called ORM.

ORM stands for Object Relational Mapping. An object-relational mapper provides an object-oriented layer between relational databases and object-oriented programming languages without having to write SQL queries.

The primary goal of ORM is to send data between a database and application models. It represents a relationship between a database and a model.

The main advantage of using Django ORM queries is that it speeds up and eliminates errors throughout the development process.

What is a QuerySet?

A Query Set is a collection of data from a database. Query set allows us to get data easily, by allowing us to filter, create, order, etc.

Creating Model:

```
from django.db import models
class  Employee(models.Model):
      eno =  models.IntegerField()
      ename = models.CharField(max_length=30)
      salary = models.FloatField()

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

Django Shell

So to enter into the Django shell, the following command should be entered into the command prompt in the virtual environment:

```
python manage.py shell
```

This will lead us to an interactive console.

```
(InteractiveConsole)
>>>
```

C - Create - Inserting data

We can insert the data in different ways like using **save()** and **create()**

save():

We take the data into one variable and we have to save the variable by using **save().**

If we don't save the variable by using **save()** then the data will not insert in the database.

When we run **save()** then automatically "INSERT INTO" command will run in the database.

create():

We use **create()** when we take the data from the user, so the variable is not required to save by using save().

create() will save the data automatically when we run the create().

When we run **create()** then automatically "INSERT INTO" command will run in the database.

Note: We goto python shell to perform CURD operations. For this execute the shell command and import the required model class name here.

E:\django7am\modelpro> python manage.py shell

>>>

>>> from modelapp.models import Employee

>>> a = Employee(eno = 10, ename = 'Virat', salary = 10000)

>>> a.save()

>>> b = Employee(eno = 20, ename = 'Rohit', salary = 30000)

>>> b.save()

>>> c = Employee.objects.create(eno = 30, ename = 'Dhoni', salary=20000)

<Employee: Dhoni>

>>> Employee.objects.create(eno = 40, ename = 'Surya', salary= 30000)

<Employee: Surya>

Note : Now goto database and check the data in Employee table

mysql > select * from employee;

==========

Id	eno	ename	salary
1	10	Virat	10000
2	20	Rohit	30000
3	30	Dhoni	20000
4	40	Surya	30000

==========

R - Retrieve - Fetching the data (get() + all(), values()...)

fetching specific value

get():

- By using get() we can get the specific condition matched element if it is available uniquely.
- If condition is not matched then it returns exception like model **DoesNotExist**
- If condition is matched with more than one time then it returns exception like **MultipleObjectsReturned**

For example,

```
>>> Employee.objects.get(id=1)
<Emp: Virat>
>>> Employee.objects.get(id=15)
FirstApp.models.Employee.DoesNotExist: Employee matching query does not exist
>>> Employee.objects.get(salary=30000)
FirstApp.models.Employee.MultipleObjectsReturned: get() returned more than one Employee -- it returned 2!
```

fetching all values

- By using **all()** we can get all records from our model table if records avaialable.
- If records not available then it returns empty queryset list like <QuerySet []>

For example,

>>> Employee.objects.all()

<QuerySet [<Employee: Virat>, <Employee: Rohit>, <Employee: Dhoni>, <Employee: Surya>]

Indexing:

- The process of accessing the specific record from QuerySet object

>>> Employee.objects.all()[0]

<Employee: Kohli>

Slicing:

- The process of accessing the group of records from QuerySet object.

>>> Employee.objects.all()[0 : 2]

<QuerySet [<Employee : Virat>, <Employee: Rohit>]

count()

- To find total available records from model table then use "count()" like below.

>>> Employee.objects.all().count()

4

values()

- It returns each object as a Python dictionary with names and values as key and value pairs respectively.

>>> Employee.objects.values()

<QuerySet [{'id': 1, 'eno': 10, 'ename': 'Virat', 'salary': 10000.0}, {'id': 2, 'eno': 20, 'ename': 'Rohit', 'salary': 30000.0}, {…..},{….}] >

>>> Employee.objects.values('id','ename')

<QuerySet [{'id': 1, 'ename': 'Kohli'}, {'id': 2, 'ename': 'Rohit'},

values_list()

>>> Employee.objects.values_list()

<QuerySet [(1, 10, 'Virat', 10000.0), (2, 20, 'Rohit', 30000.0), (…),(…)]>

>>> Employee.objects.values_list('id','ename')

<QuerySet [(1, 'Kohli'), (2, 'Rohit'), (4, 'Sami'), (5, 'Dhoni'), (6, 'Surya')]>

fllter()

- The filter() returns a filtered search. It means if given is condition matching then all matched objects returns as a QuerySet list object format

```
>>> Employee.objects.filter(id=1)
<QuerySet [ <Employee : Virat> ]>
```

- If filtered condition is not matching then it returns empty QuerySet list object.

```
>>> Employee.objects.filter(id=15)
<QuerySet [ ]>
```

exclude()

- The exclude() returns a non-filtered search. It means if given condition is matching then except condition matching elements , remaining all non matched objects returns as a QuerySet list object format.

```
>>> Employee.objects.exclude(id=1)
<QuerySet [ <Employee : Rohit> , <Employee : Dhoni> , <Employee : Surya> ]>
```

- If filtered condition is not matching then it returns empty QuerySet list object.

```
>>> Employee.objects.exclude(id=15)
<QuerySet [ ]>
```

U - Update - modifying the data

- We just assign new value to the specific field and then that variable should be saved by using **save().**
- If we don't save the variable by using save() then this modification will not perform on the database, because whenever we run save() after value is modified then internally "UPDATE" command will execute in the database.
- For example,

```
>>> emp = Employee.objects.get(id=1)
>>> emp.salary = 15000
>>> emp.save()
```

Now goto database and check the data

mysql > select * from employee;

==========

Id	eno	ename	salary
1	10	Virat	15000
2	20	Rohit	**30000**
3	30	Dhoni	20000
4	40	Surya	**30000**

==========

- Note: By using update() we can update the condition matched all records at a time also.

```
>>> Employee.objects.filter(salary=30000).update(salary=35000)
```

mysql > select * from employee;

==========

Id	eno	ename	salary
1	10	Virat	15000
2	20	Rohit	**35000**
3	30	Dhoni	20000
4	40	Surya	**35000**

==========

D - Delete - removing the data

- We get a specific record by using **get()** into a new object/ variable.
- We have to delete this new variable/object by using delete().
- If we don't use delete() to delete the object then the data will not remove from the database.
- Whenever we run delete() then internally "DELETE" command will execute in the database.

```
>>> x= Emp.objects.get(ename = 'nani')
>>> x.delete()
>>> y = Emp.objects.get(sal = 40000)
>>> y.delete()
```

- Note: By using the delete() we can delete the condition matched all records at a time.

Now check the database table

Practice: Creating data into tables

e = Employee(f1=..., f2=..., f3=....)

e.save()

Employee.objects.create(f1=...,f2=...,f3=....)

Read data from database tables

Employee.objects.all() ---->> [{},{},{},{},....]

e = Employee.objects.all()

e[1]

Employee.objects.all()[1]

indexing ---->> single record

Employee.objects.all()[0:3]

slicing ----->> group of records

Employee.objects.all().count()

count()---->> total count

Employee.objects.get(condition)

Employee.objects.filter(condition)

Employee.objects.exclude(condition)

Employee.objects.filter(condition).update(condition)

Employee.objects.filter(condition).delete()

Employee.objects.values()

Employee.objects.values('id','ename','salary')