**Django ORM**

Java: Hibernate, Spring ORM etc

**ORM--->Object Relational Mapping**

To select all employees from the employees table:

sql squery: select \* from employees

ORM-->Employee.objects.all()

To select all records:

----------------------

Employee.objects.all()

The return type of all() method is: QuerySet

emp\_list = Employee.objects.all()

print(type(emp\_list))

<class 'django.db.models.query.QuerySet'>

To get a particular record

--------------------------

We have to use get() method.

D:\djangoprojects\ormproject1>py manage.py shell

Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

(InteractiveConsole)

>>> from testapp.models import Employee

>>> emp = Employee.objects.get(id=1)

>>> emp

<Employee: Employee object (1)>

>>> emp.ename

'Gail Wilson'

>>> emp.esal

17858.0

>>> type(emp)

<class 'testapp.models.Employee'>

The return type of get() method is Employee object.

How to find query associated with QuerySet:

-------------------------------------------

Every ORM statement will be converted into sql query. We can find query from the QuerySet as follows:

>>> qs = Employee.objects.all()

>>> str(qs.query)

'SELECT "testapp\_employee"."id", "testapp\_employee"."eno", "testapp\_employee"."ename", "testapp\_employee"."esal", "testapp\_employee"."eaddr" FROM "testapp\_employee"'

How to filter records based on some condition:

----------------------------------------------

employees = Employee.objects.filter(esal\_\_gt=15000)

It returns all employees whose salary greater than 15000

employees = Employee.objects.filter(esal\_\_gte=15000)

It returns all employees whose salary greater than or equal to 15000

Similarly we can use \_\_lt and \_\_lte also.

Various possible field lookups:

-------------------------------

1. exact---> exact match

Employee.objects.get(id\_\_exact=14)

Employee.objects.get(id=14)

emp = Employee.objects.get(ename\_\_exact='Colleen Martin')

2. iexact-->case insensitive exact match.

>>> emp = Employee.objects.get(ename\_\_iexact='colleen Martin')

>>> emp.ename

'Colleen Martin'

Note: The equivalent sql query is something like

select ... where ename like 'colleen Martin'

3. contains -->case sensitive containment test

-----------

It is something like

select ... where name like '%jhon%';

emp\_list = Employee.objects.filter(ename\_\_contains='John')

4. icontains--->case insensitive containment test

--------------------------------------------------

emp\_list = Employee.objects.filter(ename\_\_contains='john')

emp\_list = Employee.objects.filter(ename\_\_icontains='john')

5. in :

------

in a given iterable like tuple or list etc

select all employees where id is in [1,3,5]

emp\_list = Employee.objects.filter(id\_\_in=[1,3,5])

6. gt:

greater than

7. gte:

-------

greater than or equal to

8. lt:

------

less than

9. lte:

--------

less than or equal to

10. startswith:

---------------

case sensitive starts with

Select all employees where ename starts with 'D'

emp\_list = Employee.objects.filter(ename\_\_startswith='D')

11. istartswith:

----------------

case insensitive starts with

12. endswith:

------------

case sensitive endswith

emp\_list = Employee.objects.filter(ename\_\_endswith='t')

emp\_list = Employee.objects.filter(ename\_\_endswith='n')

13. iendswith:

--------------

case insensitive ends with

14. range:

-----------

Range test(inclusive)

To select all employees where esal in the range 12000 to 15000

emp\_list = Employee.objects.filter(esal\_\_range=[12000,15000])

Note: check for django documentation for more field lookups.

How to implement OR Queries in Django ORM:

------------------------------------------

OR: either first condition or second condition or both

Q1. select all employees where ename starts with 'D'

emp\_list = Employee.objects.filter(ename\_\_startswith='D')

Q2. Select all employees whose salary is < 15000?

emp\_list = Employee.objects.filter(esal\_\_lt=15000)

Q3. select all employees where ename starts with 'D' or esal < 15000?

We can implement OR queries in 2 ways:

1st way:

-------

emp\_list = queryset1 | queryset2

eg:

emp\_list = Employee.objects.filter(ename\_\_startswith='D') | Employee.objects.filter(esal\_\_lt=15000)

2nd way:

--------

from django.db.models import Q

filter(Q(condition1)|Q(condition2))

eg:

emp\_list = Employee.objects.filter(Q(ename\_\_startswith='D') | Q(esal\_\_lt=15000))

summary:

1. queryset1|queryset2

2. filter(Q(condition1)|Q(condition2))

How to implement AND Queries in Django ORM:

------------------------------------------

AND: all conditions should be satisfied.

3 ways are there:

1. queryset1 & queryset2

2. filter(Q(condition1)& Q(condition2))

3. filter(condition1,condition2)

eg. select all employees where ename starts with 'D' AND esal < 15000?

1. emp\_list = Employee.objects.filter(ename\_\_startswith='D') & Employee.objects.filter(esal\_\_lt=15000)

2. emp\_list = Employee.objects.filter(Q(ename\_\_startswith='D') & Q(esal\_\_lt=15000))

3. emp\_list = Employee.objects.filter(ename\_\_startswith='D', esal\_\_lt=15000)

How to implement NOT queries in Django ORM:

-------------------------------------------

all()-->To get all records

filter(condition)-->to get records where condition is satisfied.

We can implement NOT queries in 2 ways

1st way: exclude(condition) -->to get records where condition is failed.

2nd way: filter(~Q(condition))

eg: To select all employees whose name not starts with 'J'?

emp\_list = Employee.objects.exclude(ename\_\_startswith='J')

emp\_list = Employee.objects.filter(~Q(ename\_\_startswith='J'))

Note:

get()--->To get single record

all()-->To get all records

filter()-->To get multiple records which satisfy given condition

exclude()-->To get multiple records which not satisfy given condition

How to select only required columns in the query set:

-----------------------------------------------------

select \* from employees;

select ename,esal from employees;

3 ways:

1. By using values\_list():

--------------------------

emp\_list = Employee.objects.all().values\_list('ename','esal')

<QuerySet [('Gail Wilson', 17858.0, 'Bryanfort'), ('Crystal Kim PhD', 16159.0, 'Lake Dominicport'), ('Rachel Garcia', 16595.0, 'Lopezshire'), ('Katie Jackson', 18075.0, 'Rothland'), ('Colleen Martin', 17562.0, 'Danielleport'), ('Peter Barnett', 16027.0, 'New Erika'), ('Jacob Taylor', 10822.0, 'Lake Kevinport'), ('Emily Giles', 19862.0, 'Lewishaven'), ('Danielle Miller', 13681.0, 'Emilyview'), ('Sarah Wilson', 16572.0, 'Allenport'), ('Margaret Paul', 17759.0, 'Port Jennatown'), ('Robert Fox', 10656.0, 'Port Kellychester'), ('Tony Hahn', 16631.0, 'Andrewhaven'), ('Melissa Mcclain', 13937.0, 'West Anitafort'), ('Thomas Williams', 13980.0, 'South Sandra'), ('Victoria Franklin', 16856.0, 'New Carlosville'), ('Donna Anthony', 14240.0, 'South Julietown'), ('William Galloway', 16244.0, 'Amyburgh'), ('Ashlee Mcdonald', 15432.0, 'West Davidside'), ('James Caldwell', 13881.0, 'North Timothy'), '...(remaining elements truncated)...']>

views.py:

---------

def retrieve\_view(request):

emp\_list = Employee.objects.all().values\_list('ename','esal')

return render(request,'testapp/specificcolums.html',{'emp\_list':emp\_list})

specificcolums.html

--------------------

<!DOCTYPE html>

{% extends 'testapp/base.html'%}

{% block body\_block%}

<h1>Employee Information DashBoard</h1>

<table border=2>

<thead>

<th>Employee Name</th>

<th>Employee Salary</th>

</thead>

{% for emp in emp\_list %}

<tr>

{% for v in emp %}

<td>{{v}}</td>

{% endfor %}

</tr>

{% endfor %}

</table><br>

{% endblock %}

2nd way: By using values():

---------------------------

emp\_list = Employee.objects.all().values\_list('ename','esal','eaddr')

<QuerySet [{'ename': 'Gail Wilson', 'esal': 17858.0, 'eaddr': 'Bryanfort'}, {'ename': 'Crystal Kim PhD', 'esal': 16159.0, 'eaddr': 'Lake Dominicport'}, {'ename': 'Rachel Garcia', 'esal': 16595.0, 'eaddr': 'Lopezshire'}, {'ename': 'Katie Jackson', 'esal': 18075.0, 'eaddr': 'Rothland'}, {'ename': 'Colleen Martin', 'esal': 17562.0, 'eaddr': 'Danielleport'}, {'ename': 'Peter Barnett', 'esal': 16027.0, 'eaddr': 'New Erika'}, {'ename': 'Jacob Taylor', 'esal': 10822.0, 'eaddr': 'Lake Kevinport'}, {'ename': 'Emily Giles', 'esal': 19862.0, 'eaddr': 'Lewishaven'}, {'ename': 'Danielle Miller', 'esal': 13681.0, 'eaddr': 'Emilyview'}, {'ename': 'Sarah Wilson', 'esal': 16572.0, 'eaddr': 'Allenport'}, {'ename': 'Margaret Paul', 'esal': 17759.0, 'eaddr': 'Port Jennatown'}, {'ename': 'Robert Fox', 'esal': 10656.0, 'eaddr': 'Port Kellychester'}, {'ename': 'Tony Hahn', 'esal': 16631.0, 'eaddr': 'Andrewhaven'}, {'ename': 'Melissa Mcclain', 'esal': 13937.0, 'eaddr': 'West Anitafort'}, {'ename': 'Thomas Williams', 'esal': 13980.0, 'eaddr': 'South Sandra'}, {'ename': 'Victoria Franklin', 'esal': 16856.0, 'eaddr': 'New Carlosville'}, {'ename': 'Donna Anthony', 'esal': 14240.0, 'eaddr': 'South Julietown'}, {'ename': 'William Galloway', 'esal': 16244.0, 'eaddr': 'Amyburgh'}, {'ename': 'Ashlee Mcdonald', 'esal': 15432.0, 'eaddr': 'West Davidside'}, {'ename': 'James Caldwell', 'esal': 13881.0, 'eaddr': 'North Timothy'}, '...(remaining elements truncated)...']>

views.py:

--------

def retrieve\_view(request):

emp\_list = Employee.objects.all().values('ename','esal','eaddr')

return render(request,'testapp/specificcolums.html',{'emp\_list':emp\_list})

html:

-----

<!DOCTYPE html>

{% extends 'testapp/base.html'%}

{% block body\_block%}

<h1>Employee Information DashBoard</h1>

<table border=2>

<thead>

<th>Employee Name</th>

<th>Employee Salary</th>

<th>Employee Address</th>

</thead>

{% for emp in emp\_list %}

<tr>

{% for k,v in emp.items %}

<td>{{v}}</td>

{% endfor %}

</tr>

{% endfor %}

</table><br>

{% endblock %}

3rd way:

--------

By using only()

emp\_list = Employee.objects.all().only('ename','esal','eaddr')

>>> emp\_list = Employee.objects.all().only('ename','esal','eaddr')

>>> emp\_list

<QuerySet [<Employee: Employee object (1)>, <Employee: Employee object (2)>, <Employee: Employee object (3)>, <Employee: Employee object (4)>, <Employee: Employee object (5)>, <Employee: Employee object (6)>, <Employee: Employee object (7)>, <Employee: Employee object (8)>, <Employee: Employee object (9)>, <Employee: Employee object (10)>, <Employee: Employee object (11)>, <Employee: Employee object (12)>, <Employee: Employee object (13)>, <Employee: Employee object (14)>, <Employee: Employee object (15)>, <Employee: Employee object (16)>, <Employee: Employee object (17)>, <Employee: Employee object (18)>, <Employee: Employee object (19)>, <Employee: Employee object (20)>, '...(remaining elements truncated)...']>

views.py:

---------

def retrieve\_view(request):

emp\_list = Employee.objects.all().only('ename','esal','eaddr')

return render(request,'testapp/specificcolums.html',{'emp\_list':emp\_list})

html:

-----

<!DOCTYPE html>

{% extends 'testapp/base.html'%}

{% block body\_block%}

<h1>Employee Information DashBoard</h1>

<table border=2>

<thead>

<th>Employee Name</th>

<th>Employee Salary</th>

<th>Employee Address</th>

</thead>

{% for emp in emp\_list %}

<tr>

<td>{{emp.ename}}</td>

<td>{{emp.esal}}</td>

<td>{{emp.eaddr}}</td>

</tr>

{% endfor %}

</table><br>

{% endblock %}

Note:

values\_list() -->QuerySet contains tuples

values() -->QuerySet contains dict objects

only() -->QuerySet contains Employee objects

Hence only() method is recommended to use when compared with others.

Aggregate Functions:

--------------------

Django ORM defines several functions to perform aggregate operations.

Avg(),Max(),Min(),Sum(),Count() etc

views.py:

---------

from django.db.models import Avg,Sum,Min,Max,Count

def aggregate\_view(request):

avg = Employee.objects.all().aggregate(Avg('esal'))

max = Employee.objects.all().aggregate(Max('esal'))

min = Employee.objects.all().aggregate(Min('esal'))

sum = Employee.objects.all().aggregate(Sum('esal'))

count = Employee.objects.all().aggregate(Count('esal'))

my\_dict = {'avg':avg['esal\_\_avg'],'max':max['esal\_\_max'],'min':min['esal\_\_min'],'sum':sum['esal\_\_sum'],'count':count['esal\_\_count']}

return render(request,'testapp/aggregate.html',my\_dict)

aggregate.html:

---------------

<!DOCTYPE html>

{% extends 'testapp/base.html'%}

{% block body\_block%}

<h1>Employee Aggregate Information</h1>

<ul>

<h2><li>Average Salary: {{avg}}</li></h2>

<h2><li>Maximum Salary: {{max}}</li></h2>

<h2><li>Minimum Salary: {{min}}</li></h2>

<h2><li>Total Salary: {{sum}}</li></h2>

<h2><li>Number of Employees: {{count}}</li></h2>

</ul>

{% endblock %}

How to Create,Update and Delete Records:

----------------------------------------

How to add single record:

-------------------------

1st way:

-------

e = Employee(eno=1234,ename='Dheeraj',esal=1234.0,eaddr='Bangalore')

e.save() #This employee will be inserted into database.

2nd way:

-------

Employee.objects.create(eno=7896,ename='Modi',esal=123.45,eaddr='Delhi')

How to insert/create Multiple Records at a time (Bulk Create):

------------------------------------------------

By using bulk\_create() method.

[Employee(eno=777,ename='Sachin',esal=7777,eaddr='Mumbai'),

Employee(eno=888,ename='Kohli',esal=8888,eaddr='Delhi'),

Employee(eno=999,ename='Dhoni',esal=9999,eaddr='Ranchi'),

Employee(eno=666,ename='Rohit',esal=6666,eaddr='Vizag')]

Employee.objects.bulk\_create([Employee(eno=777,ename='Sachin',esal=7777,eaddr='Mumbai'),

Employee(eno=888,ename='Kohli',esal=8888,eaddr='Delhi'),

Employee(eno=999,ename='Dhoni',esal=9999,eaddr='Ranchi'),

Employee(eno=666,ename='Rohit',esal=6666,eaddr='Vizag')])

How to delete single record:

----------------------------

e = Employee.objects.get(eno=6797)

e.delete()

How to delete multiple records:

-------------------------------

>>> qs = Employee.objects.filter(esal\_\_gte=19000)

>>> qs.count()

10

>>> qs.delete()

(10, {'testapp.Employee': 10})

>>> qs = Employee.objects.filter(esal\_\_gte=19000)

>>> qs.count()

0

How to delete all records(truncate operation in sql):

------------------------------------------------------

qs = Employee.objects.all()

qs.delete()

or

Employee.objects.all().delete()

How to update field of a Particular Record:

-------------------------------------------

e = Employee.objects.get(eno=5470)

>>> e = Employee.objects.get(eno=5470)

>>> e.eno

5470

>>> e.ename

'Derek Kennedy'

>>> e.esal

16561.0

>>> e.esal = 7777

>>> e.save()

>>> e.ename='Sunny Leone'

>>> e.save()

How to order queryset in sorting order:

---------------------------------------

emp\_list = Employee.objects.all() -->Based on insertion order

1. To sort all employees according to ascending order of eno

emp\_list = Employee.objects.all().order\_by('eno')

2. To sort all employees according to descending order of eno

emp\_list = Employee.objects.all().order\_by('-eno')

3. To get Highest Salaried employee object?

Arrange all employees in descending order and select first employee.

e = Employee.objects.all().order\_by('-esal')[0]

4. To get Second Highest Salaries Employee

e = Employee.objects.all().order\_by('-esal')[1]

5. To get all employees based on alphabetical order of names?

emp\_list = Employee.objects.all().order\_by('ename')

6. To ignore case?

By using Lower function

from django.db.models.functions import Lower

emp\_list = Employee.objects.all().order\_by(Lower('ename'))

How to perform union operation for query sets:

----------------------------------------------

By using union operation, we can combine results of 2 or more querysets from same model or from different models.

q1 = Employee.objects.filter(esal\_\_lt=15000)

q2 = Employee.objects.filter(ename\_\_startswith='J')

q3 = q1.union(q2)

To work with multiple models:

-----------------------------

Student(name,mailid,aadharnumber,marks)

Teacher(name,mailid,aadharnumber,subject,salary)

q1 = Student.objects.all().values\_list('name','mailid','aadharnumber')

q2 = Teacher.objects.all().values\_list('name','mailid','aadharnumber')

q3 = q1.union(q2)