# Chapter-11

# Django ORM

## Django ORM:

**ORM → Object Relation Mapping**

**In general we can retrive data from the database by using the following approach**


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

**Employee.objects.all()**

**The return type of all() method is QuerySet.**

```
qs=Employee.objects.all()
print(type(qs)) # <class 'django.db.models.query.QuerySet'>
```

**If we want to get only one record then we can use get() method.**

```
emp = Employee.objects.get(id=1)
print(type(emp)) #<class 'testapp.models.Employee'>
```

**The return type of get() method is Employee Object.**

## Case-1: How to find Query associated with QuerySet

```
qs = Employee.objects.all()
print(qs.query)
```

## Case-2: 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**

# Various possible Field Look ups:

### 1) exact → Exact Match

```
Entry.objects.get(id__exact=14)
```

### 2) iexact → Case-insensitive exact Match

```
Blog.objects.get(name__iexact='beatles blog')
Blog.objects.get(name__iexact=None)
```

**The equivalent queries are:**

```
SELECT ... WHERE name ILIKE 'beatles blog';
SELECT ... WHERE name IS NULL;
```

### 3) contains → Case-sensitive Containment Test

```
Entry.objects.get(headline__contains='Lennon')
```

**SQL equivalent:**

SELECT ... WHERE headline LIKE '%Lennon%';

4) icontains → Case-insensitive Containment Test

Example: Entry.objects.get(headline__icontains='Lennon')

5) in

In a given iterable; often a List, Tuple OR queryset.

Examples:

Entry.objects.filter(id__in=[1, 3, 4])

Entry.objects.filter(headline__in='abc')

6) gt

Greater than

Example: Entry.objects.filter(id__gt=4)

7) gte

Greater than or equal to.

8) lt

Less than.

9) lte

Less than or equal to.

10) startswith

Case-sensitive starts-with.

Example: Entry.objects.filter(headline__startswith='Lennon')

11) istartswith

Case-insensitive starts-with.

Example: Entry.objects.filter(headline__istartswith='Lennon')

12) endswith

Case-sensitive ends-with.

Example: Entry.objects.filter(headline__endswith='Lennon')

13) iendswith

Case-insensitive ends-with.

Example: Entry.objects.filter(headline__iendswith='Lennon')



**14) range**

**Range test (inclusive)**

```
1) import datetime
2) start_date = datetime.date(2005, 1, 1)
3) end_date = datetime.date(2005, 3, 31)
4) Entry.objects.filter(pub_date__range=(start_date, end_date))
```

**Eg-2:** employees=Employee.objects.filter(esal__range=(12000,16000))

**Note:** There are several other field lookups are possible. (Documentation)
https://docs.djangoproject.com/en/2.1/ref/models/querysets/#id4

## Case-3: How to implement OR Queries in Django ORM

2 ways are available

1) queryset_1 | queryset_2
2) filter(Q(condition1)|Q(condition2))

**Eg 1:** To get all employees whose name startswith 'A' OR salary < 15000

```
employees = Employee.objects.filter(ename__startswith='A') |
Employee.objects.filter(esal__lt=15000)
```

```
from django.db.models import Q
employees= Employee.objects.filter(Q(ename__startswith='A') | Q(esal__lt=15000))
```

## Case-4: How to implement AND Queries in Django ORM

3 ways

1) filter(condition1,condition2)
2) queryset_1 & queryset_2
3) filter(Q(condition_1)&Q(condition_2))

**Eg:** To get all employees whose name startswith 'J' AND salary < 15000

```
1) employees= Employee.objects.filter(ename__startswith='J',esal__lt=15000)
2) employees= Employee.objects.filter(ename__startswith='J') &
   Employee.objects.filter(esal__lt=15000)
3) employees= Employee.objects.filter(Q(ename__startswith='J') & Q(esal__lt=15000))
```

## Case-5: How to implement NOT Queries in Django ORM

2 ways

1) exclude(condition)
2) filter(~Q(condition))

**Eg:** To select all employees whose name not starts with 'J':

```
employees= Employee.objects.exclude(ename__startswith='J')
employees= Employee.objects.filter(~Q(ename__startswith='J'))
```

## Case-6: How to perform Union Operation for Query Sets of the Same OR different Models

By using union operation, we can combine results of 2 or more query sets.

```
q1=Employee.objects.filter(esal__lt<15000)
q2=Employee.objects.filter(ename__endswith='J')
q3=q1.union(q2)
```

**Note:** The union operator can be performed only with the querysets having the same fields and data types. Otherwise we will get error saying
django.db.utils.OperationalError: SELECTs to the left and right of UNION do not have the same number of result columns.

We can perform union operation on common columns.

**Eg:** 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)
```

## Case-7: How to select only some columns in the queryset

3 Ways

1) **By using values_list()**
   **Eg:** q1 = Student.objects.all().values_list('name','mailid','aadharnumber')

2) **By using values()**
   **Eg:** q1 = Student.objects.all().values('name','mailid','aadharnumber')

3) **By using only():**
   **Eg:** q1 = Student.objects.all().only('name','mailid','aadharnumber')

**Note:** Difference between values() and only() Methods

In the case of values() only specified columns will be selected. But in the case of only() in addition to specified columns 'id' column also will be selected.

## Case-8: Aggregate Functions

Django ORM defines several functions to perform aggregate operations.
Avg(), Max(), Min(), Sum(), Count()

**views.py**

```
1)  from django.shortcuts import render
2)  from django.db.models import Q
3)  from django.db.models import Avg,Sum,Max,Min,Count
4)  from testapp.models import Employee
5)
6)  # Create your views here.
7)  def display_view(request):
8)      avg1=Employee.objects.all().aggregate(Avg('esal'))
9)      max=Employee.objects.all().aggregate(Max('esal'))
10)     min=Employee.objects.all().aggregate(Min('esal'))
11)     sum=Employee.objects.all().aggregate(Sum('esal'))
12)     count=Employee.objects.all().aggregate(Count('esal'))
13)     my_dict={'avg':avg1,'max':max,'min':min,'sum':sum,'count':count}
14)     return render(request,'testapp/aggregate.html',my_dict)
```

**aggregate.html**

```
1)  <!DOCTYPE html>
2)  {% extends 'testapp/base.html'%}
3)      {%block body_block%}
4)      <h1>Employee Aggregate Information</h1><hr>
5)      <ul>
6)        <li>Average Salary:{{avg}}</li>
7)        <li>Max Salary:{{max}}</li>
8)        <li>Min Salary:{{min}}</li>
9)        <li>Total Salary:{{sum}}</li>
10)       <li>Number of Employees :{{count}}</li>
11)     </ul>
12)     {%endblock%}
```

## Case-9: How to Create, Update, Delete Records

**How to Add Record**

```
>>> from testapp.models import Employee
>>> Employee.objects.all().count()
30
>>> e=Employee(eno=1234,ename='Dheeraj',esal=1234.0,eaddr='Delhi')
>>> e.save()
>>> Employee.objects.all().count()
31
```

2nd Way:

```
>>> Employee.objects.create(eno=2222,ename='Sreeram',esal=10000,eaddr='Bangalore')
```

## How to Add Multiple Records at a Time:

By using bulk_create() method.

```
Employee.objects.bulk_create([Employee(eno=1,ename='DDD',esal=1000,eaddr='Hyd'),
Employee(eno=2,ename='HHH',esal=1000,eaddr='Hyd'),
Employee(eno=3,ename='MMM',esal=1000,eaddr='Hyd')])
```

## How to Delete a Single Record:

```
>>> e=Employee.objects.get(eno=1)
>>> e.eno
1
>>> e.ename
'DDD'
>>> e.delete()
```

## How to Delete Multiple Records:

```
>>> qs=Employee.objects.filter(esal__gte=15000)
>>> qs.count()
14
>>> qs.delete()
(14, {'testapp.Employee': 14})
```

## How to Delete all Records(Truncate Operation in SQL):

```
>>> Employee.objects.all().delete()
```

## How to Update Field of a Particular Record:

```
>>> from testapp.models import Employee
>>> e=Employee.objects.get(eno=7014)
```

```
>>> e.ename
'Peter Lewis'
>>> e.ename='Durga'
>>> e.save()
>>> e.ename
'Durga'
```

## How to Order queryset in Sorting Order:

```
employees = Employee.objects.all().order_by('eno')
```

All records will be arranged according to ascending order of eno
Default sorting order is ascending order

For Descending order we have to use '-'

```
employees = Employee.objects.all().order_by('-eno')
```

```
employees = Employee.objects.all().order_by('-esal')[0]
```

Returns highest salary employee object

```
employees = Employee.objects.all().order_by('-esal')[1]
```

Returns Second highest salary employee object

```
employees = Employee.objects.all().order_by('-esal')[0:3]
```

Returns list of top 3 highest salary employees info

But in the case of strings for alphabetical order:

```
employees = Employee.objects.all().order_by('ename')
```

In this case, case will be considered.
If we want to ignore case then we should use Lower() Function

```
from django.db.models.functions import Lower
employees=Employee.objects.all().order_by(Lower('ename'))
```