Functions

Sql	ORM
where	filter

OPERATERS

Relational Operators

=	=
>	__gt=
<	__lt=
>=	__gte=
<=	__lte=

Examples:

1).List the employees in dept 20

A) Emp.objects.filter(deptno=20)

2) List the employees earning more than Rs 2500

A) Emp.objects.filter(sal__gt=2500)

3) Display all salesmen

A)Emp.objects.filter(job='salesman')

Special Operators

IN		__in
LIKE	(case-in sensitive)	__startsrwith= __endswith= __contains=
ILIKE	(case-in sensitive)	__istartswith= __iendswith= __icontains=
BETWEEN		__range=(,)

IS __isnull=True/False

Examples:-

4)List the employees in dept 10 & 20

A)Emp.objects.filter(deptno__in=[10,20])

5) List all the clerks and analysts

A)Emp.objects.filter(job__in=['clerk','analyst'])

6) List all the employees whose name starts with „S"

A) Emp.objects.filter(ename__startswith='S')

2) List the employees whose name is having letter „L" as 2 nd character

A)

1) List the employees whose name is having atleast 2 L"s

2)List the employees whose name is having letter „E" as the last but one character

3) List all the employees whose name is having letter „R" in the 3 rd position

4)List all the employees who are having exactly 5 characters in their jobs

5)List the employees whose name is having atleast 5 characters.

LOGICAL OPERATORS

AND filter(Q(con) & Q(con)) or filter(, , ..)

OR | => filter(Q(con) |Q(con))

NOT filter(~Q()) or exclude()

Example:-

1) List the employees whose salary is between 200 and 300

A) Emp.objects.filter(sal__range=(2000,3000))

2**)List all the employees who don"t have a reporting manager**

A)Emp.objects.filter(manager__isnull=True)

3)List all the salesmen in dept 30

A) from django.db.models import Q

Emp.objects.filter(Q(job='SALESMAN') &Q(deptno=30))

2) List all the salesmen in dept number 30 and having salary greater than 1500

from django.db.models import Q

Emp.objects.filter(Q(job='SALESMAN') &Q(deptno=30) & Q(sal__gt=1500)

3)List all the employees whose name starts with „s" or „a"

Emp.objects.filter(Q(ename__startswith='S') | Q(ename__startswith='A'))

4) List all the employees except those who are working in dept 10 & 20.

Emp.objects.filter(~Q(deptno__in=[10,20]))

or

Emp.objects.exclude(deptno__in=[10,20])

5) List the employees whose name does not start with „S"

A) Emp.objects.filter(~Q(ename__startswith='S'))

or

Emp.objects.exclude(ename__startswith='S')

6) List all the employees who are having reporting managers in dept 10

Emp.objects.filter(manager__isnull=False ,deptno=10)

7)List the employees who are not working as managers and clerks in dept 10 and 20 with a salary in the range of 1000 to 3000

A)Emp.objects.exclude(job__in=['MANAGER','CLERK']).filter(sal__range=(1000,3000),deptno__in=[10,20])

8)List the employees whose salary not in the range of 1000 to 2000 in dept 10,20,30 except all salesmen

A)

Emp.objects.exclude(sal__range=(1000,2000),job='SALESMAN').filter(deptno__in=[10,30,20])

9) List the department names which are having letter „O" in their locations as well as their department names

A)Deptno.objects.filter(dname__contains="O",loc__contains='O').values('dname')

SORTING

.order_by('column_name') :- defaultly accending orde

.order_by('-column_name') :- deccending order by placing – before

the column_name

1).Arrange all the employees by their salary

A) Emp.objects.all().order_by('sal')

2) Arrange all the employees by their salary in the descending order

A) Emp.objects.all().order_by(‘-sal’)

3) Arrange ename, sal, job, empno and sort by descending order of salary

A)Emp.objects.values('ename','sal','job','empno').order_by('-sal')

ORDER BY should be used always as the last statement in the SQL query. I ORM also.

Selecting DISTINCT VALUES

.dinstinct()

Group Functions

We have 5 GROUP functions,

1) Sum

2) Max

3) Min

4) Avg

5) Count....etc

here we use aggregate()

we can import these functions from

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

1) display the maximum salary, minimum salary and total salary from employee

A)
Emp.objects.aggregate(min=Min('sal'),max=Max('sal'),total=Sum('sal'))

here min,max,total are alias

2) List the number of employees in department 30

Emp.objects.filter(deptno=30).count()

here count() not imported.

3)Display the total salary in department 30

Emp.objects.filter(deptno=30).aggregate(total_salary=Sum('sal'))

4) List the number of clerks in department 20

Emp.objects.filter(deptno=20,job='CLERK').count()

GROUPING

It is the process of computing the aggregates by segregating based on one or more columns. Grouping is done by using

here we use annotate()

1) Display the total salary of all departments

A) Emp.objects.values('deptno').annotate(Sum('sal'))

2) Display the maximum of each job

A) Emp.objects.values('job').annotate(Max('sal'))

here vlalues().annotate() is groping values contain attributes are grouped

HAVING

„Having" is used to filter the grouped data.

„Where" is used to filter the non grouped data.

„Having" should be used after group by clause

„Where" should be used before group by clause

1) Display job-wise highest salary only if the highest salary is more than Rs1500

A)Emp.objects.values('job').annotate(max=Max('sal')).filter(max__gt=1500)

2) Display job-wise highest salary only if the highest salary is more than 1500 excluding department

30. Sort the data based on highest salary in the ascending order.

A)Emp.objects.values('job').annotate(max=Max('sal')).filter(max__gt=1500).exclude(deptno=30).order_by('max')

RESTRICTIONS ON GROUPING

- we can select only the columns that are part of „group by" statement If

we try selecting other columns, we will get an error

1) Display the department numbers along with the number of employees in it

A)Emp.objects.values('deptno').annotate(Count('deptno'))

2) Display the department numbers which are having more than 4 employees in them

A)Emp.objects.values('deptno').annotate(c=Count('deptno')).filter(c__gt=4).order_by('deptno')

3) Display the maximum salary for each of the job excluding all the employees whose name ends with

„S"

A)Emp.objects.values('job').annotate(max=Max('sal')).exclude(ename__endswith='S')

4) Display the department numbers which are having more than 9000 as their departmental total salary

A)Emp.objects.values('deptno').annotate(total=Sum('sal')).filter(total__gt=9000)

UPDATE :-

1) Let us update salary by increasing it by Rs200 and also give commission of Rs100 where empno = 7369.

Emp.objects.filter(empno=7369).update(sal=F('sal')+200 ,cmm=100)

2) Increase all salary by 10%

A)Emp.objects.all().update(sal=F('sal')+F('sal')*10/100)

SUB – QUERIES

JOIN

1.Inner (Equi) Join:-it returns the records of two tables based on the condition like forign key relationship.

In ORM:-

select_related():-"follows" foreign-key relationships, selecting additional related-object data when it executes its query.

select_related() can applyed on forign key table.

Like

Emp.objects.select_related() :-Emp contains forign key of Dept.

It returns Emp attributes but we can call Dept attributes using reference key

or

Dept.objects.filter(emp__isnull=False) :-it is also an inner join. It returns Dept attributes

2. Left outer join :- it returns the records of left side rows which are not in another table

Dept.objects.filter(emp__isnull=True) :-

here emp is Emp Table which contain foignkey of Dept.

Database Functions

from django.db.models.functions import *

1.Comparison and conversion functions

1.Cast():-**ClassCast(expression,output_field)**

Forces the result type of **expression**to be the one from**output_field**

```
from django.db.models import FloatField
from django.db.models.functions import Cast
Emp.objects.all().annotate(float=(Cast('sal',output_field=FloatField())))
```

2.Math Functions

1.Abs :-Returns the absolute value of a numeric field or expression.

Abs(‘attribute’)

2.**Ceil:-**Returns the smallest integer greater than or equal to a numeric field or expression.

```
x_ceil=Ceil('x')
```

3.**Floor:-**Returns the largest integer value not greater than a numeric field or expression

```
x_floor=Floor('x')
```

4. Ln

Returns the natural logarithm a numeric field or expression.

```
annotate(x_ln=Ln('x'))
```

5. Log

classLog(expression1,expression2,extra)¶**

Accepts two numeric fields or expressions and returns the logarithm of the first to base of the second.

```
annotate(log=Log('x', 'y'))
```

....etc all python Math functions.

3.Text functions

1.**Chr :-**Accepts a numeric field or expression and returns the text representation of the expression as a single character. It works the same as Python's `chr()` function.

```
filter(name__startswith=Chr(ord('M')))
```

2.Concat :- joins two or more words

```
Emp.objects.all().annotate(name=Concat('ename',Value(' '),'job'))
```

here Value import from django.db.models import Value.

It used as str().

3.Left :-Returns the first **length** characters of the given text field or expression.

Left(*expression*, *length*, ***extra*)

4.Length:- it ruterns the length of the string

syntax:- name_=Length('column')