1 . **all( )** :- It returns the all attributed and rows of the Table
eg :- instance_name = table_name.objects.all( )
in SQL :- select * from table_name;

2. **get()** **:-**It returns the Only one row.we must pass unique value attributes.
Eg :- insta_name = table_name.objects.get(unique_values_attrubute= value)

note :- Default unique value attribute is ‘id’.
***.get_or_create():-** it returns single row based on the condition if it exist else it insert the data.

3. **values() :-** It returns the all dictionaries of the quaryset.The quaryset must contains multiple rows.
Eg:-instance_name = table_name.objects.all( ).values()
result :- [{'id': 1, 'name': 'mahendra', 'roll': '18F61A0625', 'ph_no': 9581921162, 'section': 'CSIT_3', 'marks': 81}]

->also we can pass attributes in the values , it returns only the passing attributes
Eg:- instance_name = table_name.objects.values(‘attribute’ , ‘ ‘, ...)
note that you can call filter(), order_by(), etc. after the values() call, that means that these two calls are identical:

```
Blog.objects.values().order_by('id')

Blog.objects.order_by('id').values()
```

4.**values_list():-**it is similar to values() but it returns in tuple
Eg:-instance_name = table_name.objects.all( ).values_list()
instance_name = table_name.objects.values_list(‘attribute’ , ‘ ‘, ...)

5.**create()** :- It is used to save the data into the table.
Eg:-instance_name = table_name.objects.create()

6. **count()** :- It returns an integer. How meny no. Of rows selected
Eg:-instance_name = table_name.objects.count()
Eg:-instance_name = table_name.objects.filter(conditin).count()

7. **lateste():-** It returns the latest object in the table based on the given field(s).
Eg:-instance_name = table_name.objects.letest(‘attribute’)

8 **first()**:- It returns the first row of the selected rowes.

Eg:-instance_name = table_name.objects.first()

Eg:-instance_name = table_name.objects.filter(condition).first()

(or)

Eg:-instance_name = table_name.objects.filter(condition).[0]

9. **last()**:- It works like first().

Eg:-instance_name = table_name.objects.last()

Eg:-instance_name = table_name.objects.filter(condition).last()

(or)

Eg:-instance_name = table_name.objects.filter(condition).[-1]

10. **aggrigate()**:-Returns a dictionary of aggregate values (averages, sums, etc.)

eg:-
```
from django.db.models import Count

q = Blog.objects.aggregate(Count('entry'))

{'entry__count': 16}

q = Blog.objects.aggregate(number_of_entries=Count('entry'))

{'number_of_entries': 16}
```

11. exists(): It returns the True (or) False.

Eg:-

```
data=Data.objects.get(id=8)

Data.objects.filter(id=data.id).exists()

True
```

12.**update()**:- it is used to update the records from the table.

Eg:-
```
Entry.objects.filter(pub_date__year=2010).update(comments_on=False)

data = Data.objects.filter(id=1)

data.update(roll='18F61A0625')
```

13 **delete():** it is used to delete the rows from the table.

Eg:- Data.objects.get(id=7).delete()

```
b = Blog.objects.get(pk=1)
```

Entry.objects.filter(blog=b).delete()

14.only():-

15 defer():-

Field lookups:-use the inside get(),filter(),exclude()...etc

1. **exact()**:-It is case-sensitive when giving the values

Eg:- Data.objects.filter(roll__exact='18f61a0625')

(it returns empty because the exat value is ‘18F61A0625’)

Data.objects.filter(roll__exact='18F61A0625') (now it returns data)

2.**iexact**:-It is not case-sensitive when giving the values

Eg:- Data.objects.filter(roll__exact='18f61a0625') (it returns the data)

Data.objects.filter(roll__exact='18F61A0625') (now it returns data)

3. **contains**:- It is case-sensitive containment test.

Eg:-Data.objects.filter(name__contains='th') (it returns data)

sql :- select * from table_name where name LIKE ‘%th%’;

4. **icontains**:- It is case-insensitive containment test.

Eg:-Data.objects.filter(name__icontains='th') (it returns data)

Data.objects.filter(name__icontains='TH') (it return same result)

sql :- select * from table_name where name ILIKE ‘%th%’;

5.**in** :- It pass the values onebyone from list (or) tuple to the condition.

Eg:- Data.objects.filter(id__in=[1,2,3,4])

6 .**gt** :- grater then (>)

gte:- grater than (or) equal (>=)

lt :- less than (<)

lte :- less than (or) equal (<=)

Eg:- Data.objects.filter(marks__lt=83).values('marks')

Data.objects.filter(marks__gt=83).values('marks')

Data.objects.filter(marks__lte=83).values('marks')

7. **startswith** :- it is case-sensitive

Eg:-Data.objects.filter(name__startswith='h')

sql :- select * from table_name where name like 'h%'

(it returns 'h' starting values in the name attribute.)

8. **istartswith** :- it is case-insensitive

Eg:-Data.objects.filter(name__startswith='H')

sql :- select * from table_name where name ilike 'h%'

(it returns 'h' starting values in the name attribute.)

9. **endswith** :- it is case-insensitive

Eg:-Data.objects.filter(name__endswith='h')

sql :- select * from table_name where name ilike '%h'

(it returns 'h' ending values in the name attribute.)

10. **istartswith** :- it is case-insensitive

Eg:-Data.objects.filter(name__iendswith='H')

(it returns 'h' endiing values in the name attribute.)

11. **range(**paramter_1,paramter_1**)** :- between in sql .

Eg:- Data.objects.filter(marks__range=(81,85))

12. date :- the passing value must be in date formate like datetime.date(year,month,day).

Eg:-Entry.objects.filter(pub_date__date=datetime.date(2005, 1, 1))

(we can use gt,gte,lt,lte along with date like bellow)

Entry.objects.filter(pub_date__date__gt=datetime.date(2005, 1, 1))

13. year :- just assign the year

Eg:-Entry.objects.filter(pub_date__date=year)

(we can use gt,gte,lt,lte along with date like bellow)

14. ***iso_year*** **:-**

15 ***month*** :-

16 .**day** :-

17. **time :-** casts the value as time.the passing value must be in date formate like datetime.time(hours,minutes).

Eg:- Entry.objects.filter(pub_date__time=datetime.time(14, 30))

18 .**isnull :-**it returns the rows based on null values in the given column.

Eg:-Data.objects.filter(name__isnull=Fal

(it returns the non null values rows of the given attribute)

Data.objects.filter(name__isnull=False)

(it returns the null value rows of the given attribute)

1.filter() :- Returns a new QuerySet containing objects that match the given lookup parameters.

In Sql :- select * from table_name where condition ;

Eg:- Data.objects.filter(section='CSIT-III')

2.**exclude()** :-Returns a new QuerySet containing objects that not match the given lookup parameters.

In Sql :- select * from table_name where not condition ;

Eg:- Data.objects.exclude(section='CSIT-III')

(returns all objects excpt section ‘CSIT-III’)

3.aggregate():-apply arthmetic operations.it gives the only aggregate columns.

from django.db.models import Count

eg :- emails_count = Store.objects.aggregate(Count('email'))

emails_count = Store.objects.aggregate(e_count=Count('email'))

here e_count aliasing the Count(‘email’)

similarlly

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

from django.db.models import Sum

from django.db.models import Variance, StdDev

3.**annotate()**:- it is used to add anothe columns .

From django.db.models import F

Eg:- Emp.objects.annotate(total_sal=F('sal')+F('cmm'))

4.**alias()** :-

5.order_by() :-It returns the rows based on the given order.

Eg:-Data.objects.order_by('id') (returns 1 to n)

Data.objects.order_by('-id') (returns n to 1)

Data.objects.filter(condition)order_by('id')

6.**reverse():-**it retuens the particular no. Of rows from the

Eg:- Data.objects.all().reverse()[:4] (returns first 4 rows)

Data.objects.all().reverse()[1:7] (returns 2 to7 rows. 1,7 indexs)

7.dates():-

8.datetimes():-

9.none():-it returns a empty list.

Eg:- Entry.objects.none()

10. **distinct() :-**

11.union():-Uses SQL’s UNION operator to combine the results of two or more QuerySets.

(both tables have same colonms.it removes duplicate rows)

Eg:- `>>> qs1.union(qs2, qs3)`

```
>>> qs1 = Author.objects.values_list('name')
>>> qs2 = Entry.objects.values_list('headline')
>>> qs1.union(qs2).order_by('name')
```

12.**interction():-**Uses SQL's instersection operator it returns trows.

(both tables have same colonms.it returns common rows)

Eg:-

```
>>> qs1 = Table.objects.all()
>>> qs2 = Table.objects.filter(name__in =['b','c'])
>>> qs1.intersection(qs2).order_by('name')
```

12.**difference():-**Uses SQL's difference operator to remove the common rows.

(both tables have same colonms.it returns common rows)

Eg:-

```
>>> qs1 = Table.objects.all()
>>> qs2 = Table.objects.filter(name__in =['b','c'])
>>> q1.difference(q2)
```

13.**select_related():-** used for one to meney retations

14.prefetch_related():- used in many to many relations

Model Functions

Q()

from django.db.models import Q

Get the Store records that have state 'CA' OR state='AZ'

Store.objects.filter(Q(state='CA') | Q(state='AZ'))

Get the Item records with name "Mocha" and "Latte"

Item.objects.filter(Q(name="Mocha") & Q(sure_name='Latte')

F(). Represents the value of a model field or annotated column.

Is used to perform operations b/w two or more attributes

from django.db.models import F

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

we can use When in annotate(),update(),aggregate(),filter()

When() Case()

from django.db.models import When ,Case

here when is if condition . If conditions is True then execute. It does not contain else part.

Case(): it allows multiple When conditions like if elif else.

Syntax:- slias=Case(When(condition , then= execute),When(),when())

When is used inside the Case()

we can use When in annotate(),update(),aggregate().

Eg:-

```
Client.objects.annotate(
...     discount=Case(
...         When(account_type=Client.GOLD, then=Value('5%')),
...         When(account_type=Client.PLATINUM, then=Value('10%')),
...         default=Value('0%'),
...     ),
... ).values_list('name', 'discount')
```

here deffault is else.

Func(). Base type for database functions like LOWER and SUM.

from django.db.models import Func

Eg:-queryset.annotate(field_lower=Func(F('field'), function='LOWER'))

here function = ‘ ’ are database functions like Text functions ,math functions ...etc

no need to import Those functions.

Value(). Expression value. Not used directly.

A **Value()** object represents the smallest possible component of an expression: a simple value. When you need to represent the value of an integer, boolean, or string within an expression, you can wrap that value within a **Value()**.

You will rarely need to use **Value()** directly. When you write the expression **F('field') + 1**, Django implicitly wraps the **1** in a **Value()**, allowing simple values to be used in more complex expressions. You will need to use **Value()** when you want to pass a string to an expression. Most expressions interpret a string argument as the name of a field, like **Lower('name')**.

ExpressionWrapper():- **it** takes two arguments expresssion and output_fiels. It create new Field.

Example adding Date filed and Time Field ,creates DateTime Field.

Eg.**from django.db.models import** DateTimeField, ExpressionWrapper, F

```
Ticket.objects.annotate(
    expires=ExpressionWrapper(
        F('active_at') + F('duration'), output_field=DateTimeField()))
```

Subquery():-

Class Subquery(queryset, output_field=None)¶

You can add an explicit subquery to a **QuerySet** using the **Subquery** expression.

```
from django.db.models import OuterRef, Subquery
    b = Deptno.objects.filter(dname='accounting')
    a =  Emp.objects.filter(deptno=Subquery(b.values('deptno')))
```

Raw SQL expressions

Class RawSQL(sql, params, output_field=None)

Sometimes database expressions can't easily express a complex **WHERE** clause. In these edge cases, use the **RawSQL** expression.

Avg():-

Count():-

Sum():-

Min():-

Max():-