Django ORM Cookbook Documentation

Release 2.0

Agiliq

Mar 26, 2018

Contents

Django ORM Cookbook is a book about doing things with Django ORM and Django models. Django is a "MTV" (Model-Template-View) framework – This book provides a deep dive into the `M` part.

They take the form of about 50 questions of the form `How to do X with Django ORM/Queryset/Models.`

How to model one to one relationships?

One-to-one relationships occur when there is exactly one record in the first table that corresponds to one record in the related table. Here we have an example where we know that each individual can have only one Biological parents i.e., Mother and Father. We already have auth user model with us, we will add a new model UserParent as described below.

```python
from django.contrib.auth.models import User

class UserParent(models.Model):
    user = models.OneToOneField(
        User,
        on_delete=models.CASCADE,
        primary_key=True,
    )
    father_name = models.CharField(max_length=100)
    mother_name = models.CharField(max_length=100)

>>> u1 = User.objects.get(first_name='Ritesh', last_name='Deshmukh')
>>> u2 = User.objects.get(first_name='Sohan', last_name='Upadhyay')
>>> p1 = UserParent(user=u1, father_name='Vilasrao Deshmukh', mother_name='Vaishali
↪Deshmukh')
>>> p1.save()
>>> p1.user.first_name
'Ritesh'
>>> p2 = UserParent(user=u2, father_name='Mr R S Upadhyay', mother_name='Mrs S K
↪Upadhyay')
>>> p2.save()
>>> p2.user.last_name
'Upadhyay'
```

The on_delete method is used to tell Django what to do with model instances that depend on the model instance you delete. (e.g. a ForeignKey relationship). The on_delete=models.CASCADE tells Django to cascade the deleting effect i.e. continue deleting the dependent models as well.

```python
>>> u2.delete()
```

Will also delete the related record of `UserParent`.

How to model one to many relationships?

In relational databases, a one-to-many relationship occurs when a parent record in one table can potentially reference several child records in another table. In a one-to-many relationship, the parent is not required to have child records; therefore, the one-to-many relationship allows zero child records, a single child record or multiple child records. To define a many-to-one relationship, use *ForeignKey*.:

```python
class Article(models.Model):
    headline = models.CharField(max_length=100)
    pub_date = models.DateField()
    reporter = models.ForeignKey(User, on_delete=models.CASCADE, related_name=
→'reporter')

    def __str__(self):
        return self.headline

    class Meta:
        ordering = ('headline',)

>>> u1 = User(username='johny1', first_name='Johny', last_name='Smith', email=
→'johny@example.com')
>>> u1.save()
>>> u2 = User(username='alien', first_name='Alien', last_name='Mars', email=
→'alien@example.com')
>>> u2.save()
>>> from datetime import date
>>> a1 = Article(headline="This is a test", pub_date=date(2018, 3, 6), reporter=u1)
>>> a1.save()
>>> a1.reporter.id
13
>>> a1.reporter
<User: johny1>
```

If you try to assign an object before saving it you will encounter a ValueError

```python
>>> u3 = User(username='someuser', first_name='Some', last_name='User', email=
→'some@example.com')
```

```
>>> Article.objects.create(headline="This is a test", pub_date=date(2018, 3, 7),␣
→reporter=u1)
Traceback (most recent call last):
...
ValueError: save() prohibited to prevent data loss due to unsaved related object
→'reporter'.
>>> Article.objects.create(headline="This is a test", pub_date=date(2018, 3, 7),␣
→reporter=u1)
>>> Article.objects.filter(reporter=u1)
<QuerySet [<Article: This is a test>, <Article: This is a test>]>
```

The above queryset shows User u1 with multiple `Articles`. Hence One to Many.

How to model many to many relationships?

A many-to-many relationship refers to a relationship between tables in a database when a parent row in one table contains several child rows in the second table, and vice versa.

Just to make it more interactive, we will talk about a twitter app. By just using few fields and ManyToMany field we can make a simple twitter app.

We basically have 3 basic things in Twitter, tweets, followers, favourite/unfavourite.

We have two models to make everything work. We are inheriting django's auth_user.:

```python
class User(AbstractUser):
    tweet = models.ManyToManyField(Tweet, blank=True)
    follower = models.ManyToManyField(settings.AUTH_USER_MODEL, blank=True)
    pass

class Tweet(models.Model):
    tweet = models.TextField()
    favourite = models.ManyToManyField(settings.AUTH_USER_MODEL, blank=True, related_
→name='user_favourite')

    def __unicode__(self):
        return self.tweet
```

What will the above model be able to do ?

```
1) User will able to follow/unfollow other users.
2) User will able to see tweets made by other users whom user is following.
3) User is able to favorite/unfavorite tweets.
```

Few operations using ManyToManyfield which can be done are:

```python
>>> t1 = Tweet(tweet="I am happy today")
>>> t1.save()
>>> t2 = Tweet(tweet="This is my second Tweet")
>>> t2.save()
>>> u1 = User(username='johny1', first_name='Johny', last_name='Smith', email=
→'johny@example.com')
```

```
>>> u1.save()
>>> u2 = User(username='johny1', first_name='Johny', last_name='Smith', email=
↪'johny@example.com')
>>> u2.save()
>>> u3 = User(username='someuser', first_name='Some', last_name='User', email=
↪'some@example.com')
>>> u3.save()
```

We have created few tweets and few users, that didn't involve any use of M2M field so far. Lets continue linking them in next step.

```
>>> u2.tweet.add(t1)
>>> u2.save()
>>> u2.tweet.add(t2)
>>> u2.save()
// User can follow other users.
>>> u2.follow.add(u1)
>>> u2.save()
// Tweets are linked to the users. Users have folloewd each other. Now we can make
↪users do favourite/unfavourite of the tweets.
>>> t1.favourite.add(u1)
>>> t1.save()
>>> t1.favourite.add(u3)
>>> t1.save()
// For removing any users vote
>>> t1.favourite.remove(u1)
>>> t1.save()
```

Working example can be found in the repo: https://github.com/yashrastogi16/simpletwitter

How to include a self-referencing Foreignkey in a model

Self-referencing fk (recursive relationship) works similar to what we do for One to Many relationships. But as the name suggests, the model references itself. Among other things, they are useful to model nested relationships.

Self reference Foreignkey can be achived in two ways.

```python
class Parent(models.Model):
    parent_ref = models.ForeignKey('self', on_delete=models.CASCADE))

# OR

class Parent(models.Model):
    parent_ref = models.ForeignKey("Parent", on_delete=models.CASCADE))
```

How to convert existing databases to Django models?

Django comes with a utility called `inspectdb` that can create models by introspecting an existing database. You can view the output by running this command

```
$ python manage.py inspectdb
```

Befor running this you will have to configure your database in the `settings.py` file. The result will be a file containing a model for each table. You may want to save that file

```
$ python manage.py inspectdb > models.py
```

The output file will be saved to your current directory. Move that file to the correct app and you have a good starting point for further customizations.

How to add a model for a database view?

A database view is a searchable object in a database that is defined by a query. Though a view doesn't store data, some refer to a views as "virtual tables," you can query a view like you can a table. A view can combine data from two or more table, using joins, and also just contain a subset of information. This makes them convenient to abstract, or hide, complicated queries.

In our SqliteStuio we can see 26 tables and no views.

Lets create a simple view.

```
create view temp_user as
    select id, first_name from auth_user;
```

After the view is created, we can see 26 tables and 1 view.

We can create its related model in our app, by `managed = False` and `db_table="temp_user"`

```
class TempUser(models.Model):
    first_name = models.CharField(max_length=100)
```

```
    class Meta:
        managed = False
        db_table = "temp_user"

// We can query the newly created view similar to what we do for any table.
>>> TempUser.objects.all().values()
<QuerySet [{'first_name': 'Yash', 'id': 1}, {'first_name': 'John', 'id': 2}, {'first_
→name': 'Ricky', 'id': 3}, {'first_name': 'Sharukh', 'id': 4}, {'first_name': 'Ritesh
→', 'id': 5}, {'first_name': 'Billy', 'id': 6}, {'first_name': 'Radha', 'id': 7}, {
→'first_name': 'Raghu', 'id': 9}, {'first_name': 'Rishabh', 'id': 10}, {'first_name
→': 'John', 'id': 11}, {'first_name': 'Paul', 'id': 12}, {'first_name': 'Johny', 'id
→': 13}, {'first_name': 'Alien', 'id': 14}]>
// You cannot insert new reord in a view.
>>> TempUser.objects.create(first_name='Radhika', id=15)
Traceback (most recent call last):
...
django.db.utils.OperationalError: cannot modify temp_user because it is a view
```

For view having union operation refer to : http://books.agiliq.com/projects/django-admin-cookbook/en/latest/database_view.html?highlight=view
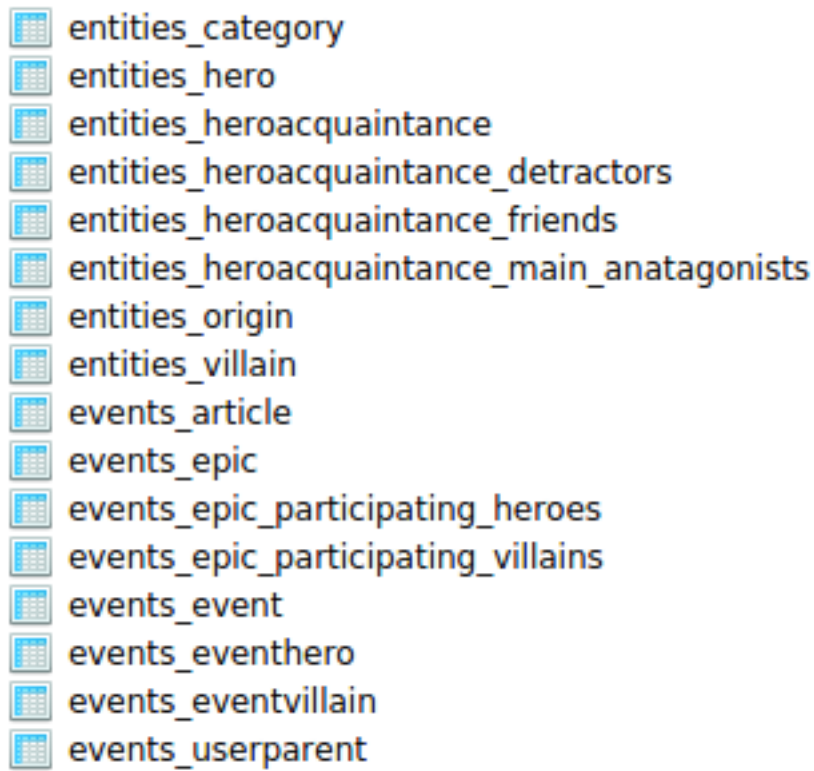
CHAPTER 7

How to model which can be related to any kind of entity? (Eg. a Tag or a
Comment?)

How to specify the table name for a model?

To save you time, Django automatically derives the name of the database table from the name of your model class and the app that contains it. A model's database table name is constructed by joining the model's "app label" – the name you used in manage.py startapp – to the model's class name, with an underscore between them.

We have two apps in our demo application i.e., `entities` and `events` so all the models in them will have app names as the prefixes followed by _ then the model name.

- entities_category
- entities_hero
- entities_heroacquaintance
- entities_heroacquaintance_detractors
- entities_heroacquaintance_friends
- entities_heroacquaintance_main_anatagonists
- entities_origin
- entities_villain
- events_article
- events_epic
- events_epic_participating_heroes
- events_epic_participating_villains
- events_event
- events_eventhero
- events_eventvillain
- events_userparent

For renaming them we cab use `db_table` parameter

```python
class TempUser(models.Model):
    first_name = models.CharField(max_length=100)
    . . .
    class Meta:
        db_table = "temp_user"
```

How to specify the column name for model field?

Naming of a column in the model can be achieved py passing a `db_column` parameter with some name. If we don't pass this parameter django creates a column with the field name which we give.

```python
class ColumnName(models.Model):
    a = models.CharField(max_length=40,db_column='column1')
    column2 = models.CharField(max_length=50)

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

▼ 📇 events_columnname
   ▼ 🗓 Columns (3)
      🗓 id
      🗓 column1
      🗓 column2
    🏷 Indexes
    ⚙ Triggers
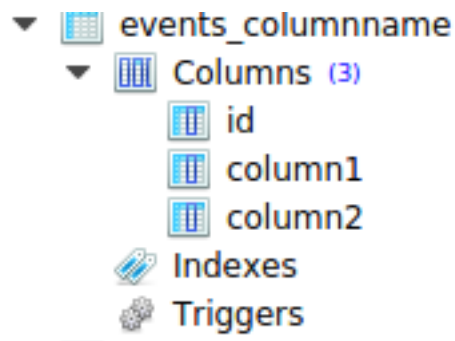
Above we can `db_coloumn` has higher priority over `field name`. First column is named as column1 but not as a.

What is the difference between `null=True` and `blank=True`?

The default value of both `null` and `blank` is `False`. Both of these values work at field level i.e., whether we want to keep a field null or blank.

`null=True` will set the field's value to NULL i.e., no data. It is basically for the databases column value.

```
date = models.DateTimeField(null=True)
```

`blank=True` determines whether the field will be required in forms. This includes the admin and your own custom forms.

```
title = models.CharField(blank=True) // title can be kept blank. In the database ("")␣
→will be stored.
```

`null=True` `blank=True` This means that the field is optional in all circumstances.

```
epic = models.ForeignKey(null=True, blank=True)
// The exception is CharFields() and TextFields(), which in Django are never saved as␣
→NULL. Blank values are stored in the DB as an empty string ('').
```

Also there is a special case, when you need to accept NULL values for a `BooleanField`, use `NullBooleanField`.

How to use a UUID instead of ID as prmary key?

Whenever we create any new model, there is an ID field attached to it. The ID field's data type will be Integer by default.

To make id field as UUID, there is a new field type UUIDField which was added in django version 1.8+.

Example

```python
import uuid
from django.db import models

class Event(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    details = models.TextField()
    years_ago = models.PositiveIntegerField()

>>> eventobject = Event.objects.all()
>>> eventobject.first().id
'3cd2b4b0c36f43488a93b3bb72029f46'
```

How to use slug field with django for more readability?

Slug is a part of a URL which identifies a particular page on a website in a form readable by users. For making it work django od=ffers us a slugfield. It can be implimented as under. We already had a model `Article` we will be adding slugfield to it to make it user readable.

```python
from django.utils.text import slugify
class Article(models.Model):
    headline = models.CharField(max_length=100)
    . . .
    slug = models.SlugField(unique=True)

    def save(self, *args, **kwargs):
        self.slug = slugify(self.headline)
        super(Article, self).save(*args, **kwargs)
    . . .

>>> u1 = User.objects.get(id=1)
>>> from datetime import date
>>> a1 = Article.objects.create(headline="todays market report", pub_date=date(2018,
→3, 6), reporter=u1)
>>> a1.save()
// slug here is auto-generated, we haven't created it in the above create method.
>>> a1.slug
'todays-market-report'
```

Slug is useful because:

> it's human friendly (eg. /blog/ instead of /1/).
> it's good SEO to create consistency in title, heading and URL.

How to add multiple databases to the django application ?

The configuration of database related stuff is mostly done in `settings.py` file. So to add multiple database to our django project we need add them in `DATABASES` dictionary.

```python
DATABASE_ROUTERS = ['path.to.DemoRouter']
DATABASE_APPS_MAPPING = {'user_data': 'users_db',
                         'customer_data':'customers_db'}

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    },
    'users_db': {
        'NAME': 'user_data',
        'ENGINE': 'django.db.backends.postgresql',
        'USER': 'postgres_user',
        'PASSWORD': 'password'
    },
    'customers_db': {
        'NAME': 'customer_data',
        'ENGINE': 'django.db.backends.mysql',
        'USER': 'mysql_cust',
        'PASSWORD': 'root'
    }
}
```

With multiple databases it will be good to talk about `Database Router`. The default routing scheme ensures that if a database isn't specified, all queries fall back to the default database. `Database Router` defaults to `[]`.

```python
class DemoRouter:
    """
    A router to control all database operations on models in the
    user application.
    """
    def db_for_read(self, model, **hints):
```

```python
        """
        Attempts to read user models go to users_db.
        """
        if model._meta.app_label == 'user_data':
            return 'users_db'
        return None

    def db_for_write(self, model, **hints):
        """
        Attempts to write user models go to users_db.
        """
        if model._meta.app_label == 'user_data':
            return 'users_db'
        return None

    def allow_relation(self, obj1, obj2, **hints):
        """
        Allow relations if a model in the user app is involved.
        """
        if obj1._meta.app_label == 'user_data' or \
           obj2._meta.app_label == 'user_data':
            return True
        return None

    def allow_migrate(self, db, app_label, model_name=None, **hints):
        """
        Make sure the auth app only appears in the 'users_db'
        database.
        """
        if app_label == 'user_data':
            return db == 'users_db'
        return None
```

Respective models would be modified as

```python
class User(models.Model):
    username = models.Charfield(ax_length=100)
    . . .
        class Meta:
        app_label = 'user_data'

class Customer(models.Model):
    name = models.TextField(max_length=100)
    . . .
        class Meta:
        app_label = 'customer_data'
```

Few helpful commands while working with multiple databases.

```
$ ./manage.py migrate --database=users_db
```

Querying and Filtering

14.1 How to find the query associated with a queryset?

We have been working with Django ORM from long time, but deep inside us sometimes comes a thought how is django ORM making our queries execute or what is the SQL of the code which we are writing.

We have a model called events for querying it with django we will write something like

```
>>> queryset = events.objects.all()
>>> print(queryset.query)
SELECT "events_event"."id", "events_event"."epic_id", "events_event"."details",
→"events_event"."years_ago" FROM "events_event"
```

```
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> queryset = Event.objects.all()
>>> print(queryset.query)
SELECT "events_event"."id", "events_event"."epic_id", "events_event"."details", "events_event"."years_ago" FROM "events_event"
>>>
```

Example 2

```
>>> queryset = Event.objects.filter(id=1, years_ago__gt=5)
>>> print(queryset.query)
SELECT "events_event"."id", "events_event"."epic_id", "events_event"."details",
→"events_event"."years_ago" FROM "events_event" WHERE ("events_event"."years_ago" >
→5 AND "events_event"."id" = 1)
```

14.2 How to do OR queries in Django ORM?

	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@aol.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com

Let's say we have a table called auth_user having fields as username, first_name, last_name, email etc.., and we want to perform AND operation on it with firstname starting with 'R' OR'ing last_name starting with 'D'.

Our SQL query for the above condition will look somethng like

```
SELECT username, first_name, last_name, email FROM auth_user WHERE first_name LIKE 'R%
→' OR last_name LIKE 'D%';
```

	username	first_name	last_name	email
1	Ricky	Ricky	Dayal	ricky@gmail.com
2	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
3	Radha	Radha	George	radha@gmail.com
4	Raghu	Raghu	Khan	raghu@rediffmail.com
5	rishab	Rishabh	Deol	rishabh@yahoo.com

Similarly our ORM query looks like

```
queryset = User.objects.filter(
        first_name__startswith='R'
    ) | User.objects.filter(
    last_name__startswith='D'
)
queryset
<QuerySet [<User: Ricky>, <User: Ritesh>, <User: Radha>, <User: Raghu>, <User: rishab>
→]>
```

You can also look at the generated query.

```
In [5]: str(queryset.query)
Out[5]: 'SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login",
→"auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name", "auth_
→user"."last_name", "auth_user"."email", "auth_user"."is_staff", "auth_user"."is_
→active", "auth_user"."date_joined" FROM "auth_user" WHERE ("auth_user"."first_name
→"::text LIKE R% OR "auth_user"."last_name"::text LIKE D%)'
```

Alternatively, you can use the `Q` objects.

```
from django.db.models import Q
qs = User.objects.filter(Q(first_name__startswith='R')|Q(last_name__startswith='D'))
```

If you look at the generated query, the result is exactly the same

```
In [9]: str(qs.query)
Out[9]: 'SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login",
→"auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name", "auth_
→user"."last_name", "auth_user"."email", "auth_user"."is_staff", "auth_user"."is_
→active", "auth_user"."date_joined" FROM "auth_user" WHERE ("auth_user"."first_name
→"::text LIKE R% OR "auth_user"."last_name"::text LIKE D%)'
```

14.3 How to do AND queries in Django ORM?

	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@aol.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com

Let's say we have a table called auth_user having fields as username, first_name, last_name, email etc.., and we want to perform OR operation on it with firstname starting with 'R' AND'ing last_name starting with 'D'.

Our SQL query for the above condition will look somethng like

```
SELECT username, first_name, last_name, email FROM auth_user WHERE first_name LIKE 'R%
→' AND last_name LIKE 'D%';
```

	username	first_name	last_name	email
1	Ricky	Ricky	Dayal	ricky@gmail.com
2	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
3	rishab	Rishabh	Deol	rishabh@yahoo.com

The default way to combine multiple conditions in `filter` is AND, so you can just do.

```
queryset_1 = User.objects.filter(
    first_name__startswith='R',
```

```
      last_name__startswith='D'
)
```

Alternatively, you can explicitly use the & operator on querysets.

```
queryset_2 = User.objects.filter(
    first_name__startswith='R'
) & User.objects.filter(
    last_name__startswith='D'
)
```

For complete customisibility, you can use the Q objects.

```
queryset_3 = User.objects.filter(
    Q(first_name__startswith='R') &
    Q(last_name__startswith='D')
)


queryset_1
<QuerySet [<User: Ricky>, <User: Ritesh>, <User: rishab>]>
```

You can look at the generated query and verify that they are all same.

```
In [10]: str(queryset_2.query)
Out[10]: 'SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login",
→"auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name", "auth_
→user"."last_name", "auth_user"."email", "auth_user"."is_staff", "auth_user"."is_
→active", "auth_user"."date_joined" FROM "auth_user" WHERE ("auth_user"."first_name
→"::text LIKE R% AND "auth_user"."last_name"::text LIKE D%)'

In [11]: str(queryset_1.query) == str(queryset_2.query) == str(queryset_3.query)
Out[11]: True
```

14.4 How to do NOT EQUAL query in Django queryset?

	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@aol.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com

Let's say we have a table called auth_user having fields as username, first_name, last_name, email etc.., and we want to perform NOT operation for fetching users with id NOT < 5.

Our SQL query for the above condition will look somethng like

```sql
SELECT id, username, first_name, last_name, email FROM auth_user WHERE NOT id < 5;
```

	id	username	first_name	last_name	email
1	5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
2	6	Billy	Billy	Sharma	billy@gmail.com
3	7	Radha	Radha	George	radha@gmail.com
4	8	sohan	Sohan	Upadhyay	sohan@aol.com
5	9	Raghu	Raghu	Khan	raghu@rediffmail.com
6	10	rishab	Rishabh	Deol	rishabh@yahoo.com

Way 1 using exclude

```
>>> queryset = User.objects.exclude(id__lt=5)
>>> queryset
<QuerySet [<User: Ritesh>, <User: Billy>, <User: Radha>, <User: sohan>, <User: Raghu>,
→ <User: rishab>]>
```

Way 2 using Q() method aka query tools

```
>>> from django.db.models import Q
>>> queryset = User.objects.filter(~Q(id__lt=5))
>>> queryst
<QuerySet [<User: Ritesh>, <User: Billy>, <User: Radha>, <User: sohan>, <User: Raghu>,
→ <User: rishab>]>
```

14.5 How to do union of two querysets from same or different models?

The UNION operator is used to combine the result-set of two or more querysets.

Lets continue with our auth_user model and generate 2 querysets to perform union operation

```
>>> q1 = User.objects.filter(id__gte=5)
>>> q1
<QuerySet [<User: Ritesh>, <User: Billy>, <User: Radha>, <User: sohan>, <User: Raghu>,
→ <User: rishab>]>
>>> q2 = q2 = User.objects.filter(id__lte=9)
>>> q2
<QuerySet [<User: yash>, <User: John>, <User: Ricky>, <User: sharukh>, <User: Ritesh>,
→ <User: Billy>, <User: Radha>, <User: sohan>, <User: Raghu>]>
>>> q1.union(q2)
<QuerySet [<User: yash>, <User: John>, <User: Ricky>, <User: sharukh>, <User: Ritesh>,
→ <User: Billy>, <User: Radha>, <User: sohan>, <User: Raghu>, <User: rishab>]>
>>> q2.union(q1)
<QuerySet [<User: yash>, <User: John>, <User: Ricky>, <User: sharukh>, <User: Ritesh>,
→ <User: Billy>, <User: Radha>, <User: sohan>, <User: Raghu>, <User: rishab>]>
```

Now try this

```
>>> q3 = EventVillain.objects.all()
>>> q3
<QuerySet [<EventVillain: EventVillain object (1)>]>
>>> q1.union(q3)
django.db.utils.OperationalError: SELECTs to the left and right of UNION do not have␣
↪the same number of result columns
```

The union operation can be performed only with the querysets having same fields and the datatypes. Hence our last union operation encountered error. You can do a union on two models as long as they have same fields or same subset of fields.

Since `Hero` and `Villain` both have the `name` and `gender`, we can use `values_list` to limit the selected fields then do a union.

```
Hero.objects.all().values_list("name", "gender").union(Villain.objects.all().values_
↪list("name", "gender"))
```

14.6 How to select some fields only in a queryset?

	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@aol.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com

We have a auth_user model having some 5 fields in it. But sometimes we are not in need to use all the fields so for such situations we can query only desired fields. This can be achieved my `values` method.

Say, we want to get firstname and lastname off all the users having initials as **R**

```
>>> User.objects.filter(first_name__startswith='R').values('first_name', 'last_name')
<QuerySet [{'first_name': 'Ricky', 'last_name': 'Dayal'}, {'first_name': 'Ritesh',
↪'last_name': 'Deshmukh'}, {'first_name': 'Radha', 'last_name': 'George'}, {'first_
↪name': 'Raghu', 'last_name': 'Khan'}, {'first_name': 'Rishabh', 'last_name': 'Deol'}
↪]
```

The output will be list of dictionaries with desired result.

14.7 How to do a subquery experession in Django?

Subquery expression which is included in `Django` from version `1.11`. Lets talk about this with an example. We have a UserParent table which has OnetoOne relation with aut user. We will find all the users having their parents detail saved.

```
>>> users = User.objects.all()
>>> UserParent.objects.filter(user_id__in=Subquery(users.values('id')))
<QuerySet [<UserParent: UserParent object (2)>, <UserParent: UserParent object (5)>,
→<UserParent: UserParent object (8)>]>
```

14.8 How to filter a queryset with criteria based on comparing their field values

Django ORM makes it easy to filter based on fixed values. To get all `User` objects with `first_name` starting with :code:''R', *you can do :code:'User.objects.filter(name_startswith='R').*

What if you want to compare the first_name and last name? You can use the `F` object. Create some users first.

```
In [27]: User.objects.create_user(email="shabda@example.com", username="shabda",␣
→first_name="Shabda", last_name="Raaj")
Out[27]: <User: shabda>

In [28]: User.objects.create_user(email="guido@example.com", username="Guido", first_
→name="Guido", last_name="Guido")
Out[28]: <User: Guido>
```

Now you can find the users where `first_name==last_name`

```
In [29]: User.objects.filter(last_name=F("first_name"))
Out[29]: <QuerySet [<User: Guido>]>
```

`F` also works with calculated field using annotate. What if we wanted users whose first and last names have same letter?

You can set the first letter from a string using `Substr("first_name", 1, 1)`, so we do.

```
In [41]: User.objects.create_user(email="guido@example.com", username="Tim", first_
→name="Tim", last_name="Teters")
Out[41]: <User: Tim>
#...
In [46]: User.objects.annotate(first=Substr("first_name", 1, 1), last=Substr("last_
→name", 1, 1)).filter(first=F("last"))
Out[46]: <QuerySet [<User: Guido>, <User: Tim>]>
```

`F` can also be used with `__gt`, `__lt` and other expressions.

14.9 How to filter FileField without any file?

Any `FileField` or `ImageField` mostly stores the path of the image wherever they are stored. So at sometimes when images/files are not stored the `Field` will be left blank as a empty string. So to query FileField without any file we can query as under.

```
no_files_objects = MyModel.objects.filter(file='') // This query will return all the
→objects of the model which don't have file stored in them.
```

14.10 How to perform join operations in django ORM?

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Join can be carried out in many ways. Some are shown below.

```
>>> a1 = Article.objects.select_related('reporter') // Using select_related
>>> a1
<QuerySet [<Article: International News>, <Article: Local News>, <Article: Morning␣
→news>, <Article: Prime time>, <Article: Test Article>, <Article: Weather Report>]>
>>> print(a1.query)
SELECT "events_article"."id", "events_article"."headline", "events_article"."pub_date
→", "events_article"."reporter_id", "events_article"."slug", "auth_user"."id", "auth_
→user"."password", "auth_user"."last_login", "auth_user"."is_superuser", "auth_user".
→"username", "auth_user"."first_name", "auth_user"."last_name", "auth_user"."email",
→"auth_user"."is_staff", "auth_user"."is_active", "auth_user"."date_joined" FROM
→"events_article" INNER JOIN "auth_user" ON ("events_article"."reporter_id" = "auth_
→user"."id") ORDER BY "events_article"."headline" ASC
>>> a2 = Article.objects.filter(reporter__username='John')
>>> a2
<QuerySet [<Article: International News>, <Article: Local News>, <Article: Prime time>
→, <Article: Test Article>, <Article: Weather Report>]>
>>> print(a2.query)
SELECT "events_article"."id", "events_article"."headline", "events_article"."pub_date
→", "events_article"."reporter_id", "events_article"."slug" FROM "events_article"␣
→INNER JOIN "auth_user" ON ("events_article"."reporter_id" = "auth_user"."id") WHERE
→"auth_user"."username" = John ORDER BY "events_article"."headline" ASC
```

14.11 How to find second largest record using Django ORM ?

We have come across situations when we want to query second highest user depending on his age or salary. Though ORM gives as the flexibility of finding first(), last() item from the queryset but not nth item.

	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@aol.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com

We can find Nth records from the query by using slice operator.

```
>>> u = User.objects.order_by('-id')[1] // Second Highest record w.r.t 'id'
>>> u.first_name
'Raghu'
>>> u = User.objects.order_by('-id')[2] // Third Highest record w.r.t 'id'
>>> u.first_name
'Sohan'
```

14.12 Find rows which have duplicate field values

id	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@gmail.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com
11	johny	John	Smith	john@example.com
12	paul	Paul	Jones	paul@example.com
13	johny1	John	Smith	johny@example.com

We can find duplicate records from the query as under.:

```
>>> duplicates = User.objects.values('first_name').annotate(Count('id')).order_by().
→filter(id__count__gt=1)
>>> duplicates
<QuerySet [{'first_name': 'John', 'id__count': 3}]>
>>> records = User.objects.filter(first_name__in=[item['first_name'] for item in
→duplicates])
>>> print([item.id for item in records])
[2, 11, 13]
```

14.13 How to find distinct field values from queryset?

id	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@gmail.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com
11	johny	John	Smith	john@example.com
12	paul	Paul	Jones	paul@example.com
13	johny1	John	Smith	johny@example.com

We can find distinct records from the query as under.:

```
>>> distinct = User.objects.values('first_name').annotate(Count('id')).order_by().
→filter(id__count=1)
>>> distinct
<QuerySet [{'id__count': 1, 'first_name': 'Billy'}, {'id__count': 1, 'first_name':
→'Paul'}, {'id__count': 1, 'first_name': 'Radha'}, {'id__count': 1, 'first_name':
→'Raghu'}, {'id__count': 1, 'first_name': 'Ricky'}, {'id__count': 1, 'first_name':
→'Rishabh'}, {'id__count': 1, 'first_name': 'Ritesh'}, {'id__count': 1, 'first_name
→': 'Sharukh'}, {'id__count': 1, 'first_name': 'Sohan'}, {'id__count': 1, 'first_name
→': 'Yash'}]>
>>> records = User.objects.filter(first_name__in=[item['first_name'] for item in
→distinct])
>>> print([item.id for item in records])
[1, 3, 4, 5, 6, 7, 8, 9, 10, 12]
```

14.14 How to solve problems using query related tools?

In our previous examples when we did OR'ing and AND'ing of the queries. Similar queries could be done using
Query related tools

```
>>> from django.db.models import Q
>>> queryset = User.objects.filter(Q(first_name__startswith='R') | Q(last_name__
→startswith='D')) // OR Operation
>>> queryset
<QuerySet [<User: Ricky>, <User: Ritesh>, <User: Radha>, <User: Raghu>, <User: rishab>
→]>
>>> queryset = User.objects.filter(Q(first_name__startswith='R') & Q(last_name__
→startswith='D')) // AND Operation
```

```
>>> queryset
<QuerySet [<User: Ricky>, <User: Ritesh>, <User: rishab>]>
```

Some other query related tools apart from `Q()` objects are `select_related()` and `prefetch_related()`.

14.15 How to group records in Django ORM?

Grouping of records in Django ORM can be done using aggregation functions like `Max`, `Min`, `Avg`, `Sum`. Django queries help to create, retrieve, update and delete objects. But sometimes we need to get aggregated values from the objects. We can get them by example shown below

```
>>> from django.db.models import Avg, Max, Min, Sum, Count
>>> User.objects.all().aggregate(Avg('id'))
{'id__avg': 7.571428571428571}
>>> User.objects.all().aggregate(Max('id'))
{'id__max': 15}
>>> User.objects.all().aggregate(Min('id'))
{'id__min': 1}
>>> User.objects.all().aggregate(Sum('id'))
{'id__sum': 106}
```

14.16 How to convert string to datetime and store in database?

We can convert a date-string and store it in the database using django in many ways. Few of them are discussed below. Lets say we have a date-string as "2018-03-11" we can not directly store it to our date field, so we can use some dateparser or python library for it.

```
>>> user = User.objects.get(id=1)
>>> date_str = "2018-03-11"
>>> from django.utils.dateparse import parse_date // Way 1
>>> temp_date = parse_date(date_str)
>>> a1 = Article(headline="String converted to date", pub_date=temp_date,␣
→reporter=user)
>>> a1.save()
>>> a1.pub_date
datetime.date(2018, 3, 11)
>>> from datetime import datetime // Way 2
>>> temp_date = datetime.strptime(date_str, "%Y-%m-%d").date()
>>> a2 = Article(headline="String converted to date way 2", pub_date=temp_date,␣
→reporter=user)
>>> a2.save()
>>> a2.pub_date
datetime.date(2018, 3, 11)
```

14.17 How to efficiently select a random object from a model?

Your `category` models is like this.

```
class Category(models.Model):
    name = models.CharField(max_length=100)
```

```
    class Meta:
        verbose_name_plural = "Categories"

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

You want to get a random Category. We will look at few alternate ways to do this.

The most straightforward way, you can order_by random and fetch the first record. It would look something like this.

```
def get_random():
    return Category.objects.order_by("?").first()
```

Note: `order_by('?')` queries may be expensive and slow, depending on the database backend you're using. To test other methods, we need to instert one million records in `Category` table. Go to your db like with `python manage.py dbshell` and run this.

```
INSERT INTO entities_category
            (name)
(SELECT Md5(Random() :: text) AS descr
 FROM   generate_series(1, 1000000));
```

You don't need to understand the full details of the sql above, it creates one million numbers and `md5-s` them to generate the name, then inserts it in the DB.

Now, instead of sorting the whole table, you can get the max id, generate a random number in range [1, max_id], and filter that. You are assuming that there have been no deletions.

```
In [1]: from django.db.models import Max

In [2]: from entities.models import Category

In [3]: import random

In [4]: def get_random2():
   ...:     max_id = Category.objects.all().aggregate(max_id=Max("id"))['max_id']
   ...:     pk = random.randint(1, max_id)
   ...:     return Category.objects.get(pk=pk)
   ...:

In [5]: get_random2()
Out[5]: <Category: e2c3a10d3e9c46788833c4ece2a418e2>

In [6]: get_random2()
Out[6]: <Category: f164ad0c5bc8300b469d1c428a514cc1>
```

If your models has deletions, you can sligthtly modify the functions, to loop until you get a valid `Category`.

```
In [8]: def get_random3():
   ...:     max_id = Category.objects.all().aggregate(max_id=Max("id"))['max_id']
   ...:     while True:
   ...:         pk = random.randint(1, max_id)
   ...:         category = Category.objects.filter(pk=pk).first()
   ...:         if category:
   ...:             return category
   ...:
```

```
In [9]: get_random3()
Out[9]: <Category: 334aa9926bd65dc0f9dd4fc86ce42e75>

In [10]: get_random3()
Out[10]: <Category: 4092762909c2c034e90c3d2eb5a73447>
```

Unless your model has a lot of deletions, the `while True:` loop return quickly. Lets use `timeit` to see the differences.

```
In [14]: timeit.timeit(get_random3, number=100)
Out[14]: 0.20055226399563253

In [15]: timeit.timeit(get_random, number=100)
Out[15]: 56.92513192095794
```

`get_random3` is about 283 time faster than `get_random`. `get_random` is the most generic way, but the technique in `get_random3` will work unless you change changed the default way Django generates the id - autoincrementing integers, or there have been too many deletions.

14.18 How to use arbitrary database functions in querysets?

Django comes with functions like `Lower`, `Coalesce` and `Max`, but it can't support all database functions, expecially ones which are database specific.

Django provides `Func` which allows using arbitrary database functions, even if Django doesn't provide them.

Postgres has `fuzzystrmatch`, which provides several functions to determine similarities. Install; the extension in your postgres DB with `create extension fuzzystrmatch`

We will use the `levenshtein` function. Lets first create some Hero objects.

```
Hero.objects.create(name="Zeus", description="A greek God", benevolence_factor=80,
→category_id=12, origin_id=1)
Hero.objects.create(name="ZeuX", description="A greek God", benevolence_factor=80,
→category_id=12, origin_id=1)
Hero.objects.create(name="Xeus", description="A greek God", benevolence_factor=80,
→category_id=12, origin_id=1)
Hero.objects.create(name="Poseidon", description="A greek God", benevolence_factor=80,
→ category_id=12, origin_id=1)
```

We want to find out the `Hero` objects which have `name` similar to Zeus. You can do

```
from django.db.models import Func, F
Hero.objects.annotate(like_zeus=Func(F('name'), function='levenshtein', template="
→%(function)s(%(expressions)s, 'Zeus')"))
```

The `like_zeus=Func(F('name'), function='levenshtein', template="%(function)s(%(expressions)s, 'Zeus')")` took two arguments which allowed the database representation, viz, `function` and `template`. If you need to reuse the function, you can define a class like this.

```
class LevenshteinLikeZeus(Func):
    function='levenshtein'
    template="%(function)s(%(expressions)s, 'Zeus')"
```

And then use `Hero.objects.annotate(like_zeus=LevenshteinLikeZeus(F("name")))`

You can then filter on this annotated field like this.

```
In [16]: Hero.objects.annotate(
    ...:         like_zeus=LevenshteinLikeZeus(F("name"))
    ...:     ).filter(
    ...:         like_zeus__lt=2
    ...:     )
    ...:
Out[16]: <QuerySet [<Hero: Zeus>, <Hero: ZeuX>, <Hero: Xeus>]>
```

Ordering things

15.1 How to order a queryset in ascending or descending order?

Ordering of the queryset can be achieved by `order_by` method. We need to pass the field on which we need to Order (ascending/descending) the result. Query looks like this

```
>>> User.objects.all().order_by('date_joined') // For ascending
<QuerySet [<User: yash>, <User: John>, <User: Ricky>, <User: sharukh>, <User: Ritesh>,
↪ <User: Billy>, <User: Radha>, <User: Raghu>, <User: rishab>, <User: johny>, <User:␣
↪paul>, <User: johny1>, <User: alien>]>
>>> User.objects.all().order_by('-date_joined') // For descending; Not '-' sign in␣
↪order_by method
<QuerySet [<User: alien>, <User: johny1>, <User: paul>, <User: johny>, <User: rishab>,
↪ <User: Raghu>, <User: Radha>, <User: Billy>, <User: Ritesh>, <User: sharukh>,
↪<User: Ricky>, <User: John>, <User: yash>]>
```

15.2 How to order a queryset in case insensitive manner?

id	username	first_name	last_name	email
1	yash	Yash	Rastogi	yashr@agiliq.com
2	John	John	Kumar	john@gmail.com
3	Ricky	Ricky	Dayal	ricky@gmail.com
4	sharukh	Sharukh	Misra	sharukh@hotmail.com
5	Ritesh	Ritesh	Deshmukh	ritesh@yahoo.com
6	Billy	Billy	Sharma	billy@gmail.com
7	Radha	Radha	George	radha@gmail.com
8	sohan	Sohan	Upadhyay	sohan@gmail.com
9	Raghu	Raghu	Khan	raghu@rediffmail.com
10	rishab	Rishabh	Deol	rishabh@yahoo.com
11	johny	John	Smith	john@example.com
12	paul	Paul	Jones	paul@example.com
13	johny1	John	Smith	johny@example.com

Whenever we try to do `order_by` with some string value, the ordering happens alphabetically and w.r.t case. Like

```
>>> User.objects.all().order_by('username').values_list('username', flat=True)
<QuerySet ['Billy', 'John', 'Radha', 'Raghu', 'Ricky', 'Ritesh', 'johny', 'johny1',
→'paul', 'rishab', 'sharukh', 'sohan', 'yash']>
```

If we want to order queryset in case insensitive manner, we can do like this .

```
>>> from django.db.models.functions import Lower
>>> User.objects.all().order_by(Lower('username')).values_list('username', flat=True)
<QuerySet ['Billy', 'John', 'johny', 'johny1', 'paul', 'Radha', 'Raghu', 'Ricky',
→'rishab', 'Ritesh', 'sharukh', 'sohan', 'yash']>
```

15.3 How to order on two fields

`order_by` on querysets can take one or more attrinute names, allowing you to order on two or more fields.

..code-block:: ipython

In [5]: from django.contrib.auth.models import User

In [6]: User.objects.all().order_by("is_active", "-last_login", "first_name") Out[6]: <QuerySet [<User: Guido>, <User: shabda>, <User: Tim>]>

15.4 How to order on a field from a related model (with a foreign key)?

15.5 How to order on an annotated field?

Here we will talk about two models i.e., `Article` and `User`. Article has FK relationship with User model. Here we will be counting the articles created by user and order them in ascending order.

```
>>> from django.db.models import Sum
>>> Article.objects.values('reporter__username').annotate(reporter_article = Sum(
↪'reporter')).order_by('reporter_article')
<QuerySet [{'reporter_article': 2, 'reporter__username': 'yash'},
 {'reporter_article': 10, 'reporter__username': 'John'},
  {'reporter_article': 11, 'reporter__username': 'johny'}]>
```

Creating, Updating and Deleting things

16.1 How to create multiple objects in one shot?

There are conditions when we want to save multiple objects in one go. Say we want to add multiple categories at once and we don't want to make many queries to the database. We can use `bulk_create` for creating multiple objects in one shot.

Here is an example.

```
>>> Category.objects.all().count()
2
>>> Category.objects.bulk_create([Category(name="Foods Items"), Category(name="Fruits
↪"), Category(name="Vegetables")])
[<Category: Foods Items>, <Category: Fruits>, <Category: Vegetables>]
>>> Category.objects.all().count()
5
```

16.2 How to copy or clone an existing model object?

There is no built-in method for copying model instances, it is possible to easily create new instance with all fields' values copied. That instance can be saved again by setting instance's `pk` to `None`. For example

```
>>> Article.objects.all().count()
5
>>> reporter = User.objects.get(id = 1)
>>> reporter
<User: yash>
>>> article = Article(headline="Article to test copying instance", pub_date=date(2018,
↪ 3, 11), reporter=reporter)
>>> article.pk
7
>>> article.pk = None // Assigning None to pk, to make a copy of an instance.
```

```
>>> article.save()
>>> article.pk
8
>>> Article.objects.all().count()
7
```

16.3 How to ensure that only one object can be created?

16.4 How to update denormalized fields in other models on save?

16.5 How to perform truncate like operation using Django ORM?

Truncate statement in SQL is mainly meant to deallocate a table, i.e., empty a table for future use. Though django doesn't provide any such operation https://code.djangoproject.com/ticket/16427, but still similar result can be achived using `delete()` method. For example:

```
>>> Article.objects.all().count()
7
>>> Article.objects.all().delete()
(7, {'events.Article': 7})
>>> Article.objects.all().count()
0
```

From above example it is clear that Article model has no data in it. Which was expected from Truncate statement.

16.6 What signals are raised by Django during object creation or up-dation?

Testing

17.1 How to assert that a function used a fixed number of queries?

We can count number of queries for testing by using `assertNumQueries()` method.

```python
def test_number_of_queries(self):
    User.objects.create(username='testuser1', first_name='Test', last_name='user1')
    # Above ORM create will run only one query.
    self.assertNumQueries(1)
    User.objects.filter(username='testuser').update(username='test1user')
    # One more query added.
    self.assertNumQueries(2)
```

17.2 How to speed tests by reusing database between test runs?

When we execute the command `python manage.py test`, a new db is created everytime. This doesn't matter much if we don't have many migrations.

But when we have many migrations, it takes a long time to recreate the database between the test runs. To avoid such situations, we may reuse the old database.

You can prevent the test databases from being destroyed by adding the `--keepdb` flag to the test command. This will preserve the test database between runs. If the database does not exist, it will first be created. If any migrations have been added since the last test run, they will be applied in order to keep it up to date.

```
$ python manage.py test --keepdb
```

17.3 How to reload a model object from the database?

Models can be reloaded from the databse using `refresh_from_db()` method. THis proves helpful during testing. For example.

```python
class TestORM(TestCase):
def test_update_result(self):
    userobject = User.objects.create(username='testuser', first_name='Test', last_
→name='user')
    User.objects.filter(username='testuser').update(username='test1user')
    # At this point userobject.val is still testuser, but the value in the database
    # was updated to test1user. The object's updated value needs to be reloaded
    # from the database.
    userobject.refresh_from_db()
    self.assertEqual(userobject.username, 'test1user')
```

17.4 How to provide initial/test data to tests?

CHAPTER 18

Indices and tables

- genindex
- modindex
- search