References


Most of the examples in this document are from the above references

Reading


One-Many

A poll can have many choices

A choice belongs to one poll

class Poll(models.Model):
    question = models.CharField(maxlength=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(maxlength=200)
    votes = models.IntegerField()
## One-to-Many Table

### polls_poll

<table>
<thead>
<tr>
<th>id</th>
<th>question</th>
<th>pub_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Do you like Django Reinhardt's music?</td>
<td>2006-09-13 18:45</td>
</tr>
<tr>
<td>2</td>
<td>Do you like Python?</td>
<td>2006-09-14 11:22</td>
</tr>
</tbody>
</table>

### polls_choice

<table>
<thead>
<tr>
<th>id</th>
<th>poll_id</th>
<th>choice</th>
<th>votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Yes</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Who is Django Reinhardt?</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Yes</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>No</td>
<td>0</td>
</tr>
</tbody>
</table>
Many-to-Many

An author can have many books

class Author(models.Model):
    name = models.CharField(maxlength=50)

A book can have many authors

class Book(models.Model):
    name = models.CharField(maxlength=50)
    authors = models.ManyToManyField(Author)
Many-to-Many Table Creation

->manage.py sqlall books
BEGIN;
CREATE TABLE "books_book" (  
    "id" integer NOT NULL PRIMARY KEY,  
    "name" varchar(50) NOT NULL
);
CREATE TABLE "books_author" (  
    "id" integer NOT NULL PRIMARY KEY,  
    "name" varchar(50) NOT NULL
);
CREATE TABLE "books_book_authors" (  
    "id" integer NOT NULL PRIMARY KEY,  
    "book_id" integer NOT NULL REFERENCES "books_book" ("id"),  
    "author_id" integer NOT NULL REFERENCES "books_author" ("id"),  
    UNIQUE ("book_id", "author_id")
);
COMMIT;
### Many-to-Many Tables

**books_book**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pro Django: Web Development</td>
</tr>
<tr>
<td></td>
<td>Done Right</td>
</tr>
</tbody>
</table>

**books_author**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adrian Holovaty</td>
</tr>
<tr>
<td>2</td>
<td>Jacob Kaplan-Moss</td>
</tr>
</tbody>
</table>

**books_book_authors**

<table>
<thead>
<tr>
<th>id</th>
<th>book_id</th>
<th>author_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
python manage.py shell

```python
>>> from cs683.books.models import *
>>> django = Book.objects.get(id=1)
>>> django
<Book: Pro Django: Web Development Done Right>

>>> django.authors.all()
[<Author: Adrian Holovaty>, <Author: Jacob Kaplan-Moss>]

>>> adrian = Author.objects.get(id=1)
>>> adrian
<Author: Adrian Holovaty>
>>> adrian.book_set.all()
[<Book: Pro Django: Web Development Done Right>]
```
Renaming the Access

```python
class Book(models.Model):
    name = models.CharField(maxlength=50)
    authors = models.ManyToManyField(Author, related_name='books')

>>> adrian = Author.objects.get(id=1)

>>> adrian.books.all()
[<Book: Pro Django: Web Development Done Right>]

>>> adrian.book_set.all()
Traceback (most recent call last):
  File "<console>", line 1, in ?
AttributeError: 'Author' object has no attribute 'book_set'
```
Many-to-Many with Extra Data

Article ← Writer of position → Reporter
from django.db import models

class Reporter(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)

class Article(models.Model):
    headline = models.CharField(max_length=100)
    pub_date = models.DateField()

class Writer(models.Model):
    reporter = models.ForeignKey(Reporter)
    article = models.ForeignKey(Article)
    position = models.CharField(max_length=100)
One-to-One

"The semantics of one-to-one relationships will be changing soon, so we don't recommend you use them"
class Book(models.Model):
    name = models.CharField(maxlength=50)
    authors = models.ManyToManyField(Author, related_name='books')

class ScienceBook(Book):
    subject = models.CharField(maxlength=50)

Would like but don't get:
    ScienceBook to inherit authors
    Book.objects.all() to return Book and ScienceBook objects
The Object-Relational Impedence Mismatch

Objects are different than tables

Relating to self

```python
class Person(models.Model):
    full_name = models.CharField(maxlength=20)
    mother = models.ForeignKey('self', null=True, related_name='mothers_child_set')
    father = models.ForeignKey('self', null=True, related_name='fathers_child_set')

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

class Person(models.Model):
    name = models.CharField(maxlength=20)
    friends = models.ManyToManyField('self')
    idols = models.ManyToManyField('self', symmetrical=False, related_name='stalkers')

    def __str__(self):
        return self.name
```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(maxlength=20)
    last_name = models.CharField(maxlength=20)

    def __str__(self):
        return "%s %s" % (self.first_name, self.last_name)

    def save(self):
        print "Before save"
        super(Person, self).save() # Call the "real" save() method
        print "After save"

    def delete(self):
        print "Before deletion"
        super(Person, self).delete() # Call the "real" delete() method
        print "After deletion"
from django.db import models

GENDER_CHOICES = (
    ('M', 'Male'),
    ('F', 'Female'),
)

class Person(models.Model):
    name = models.CharField(max_length=20)
    gender = models.CharField(max_length=1, choices=GENDER_CHOICES)

    def __str__(self):
        return self.name
get display

```python
>>> a = Person(name='Adrian', gender='M')
>>> a.save()
>>> s = Person(name='Sara', gender='F')
>>> s.save()
>>> a.gender
'M'
>>> s.gender
'F'
>>> a.get_gender_display()
'Male'
>>> s.get_gender_display()
'Female'
```
from django.db.models import Q

Article.objects.filter(headline__startswith='Hello') | Article.objects.filter(headline__startswith='Goodbye')

Article.objects.filter(Q(headline__startswith='Hello') | Q(headline__startswith='Goodbye'))

Article.objects.filter(Q(headline__startswith='Hello') & Q(headline__startswith='Goodbye'))