CS 683 Emerging Technologies
Fall Semester, 2006
Doc 7 Django Database
Sep 19, 2006

References

Django Model Examples, http://www.djangoproject.com/documentation/models/

Django Database API reference,http://www.djangoproject.com/documentation/db_api/

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

Reading

Django Database API reference,http://www.djangoproject.com/documentation/db_api/

Django Model Reference, http://www.djangoproject.com/documentation/model_api/

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

id	question	pub_date
1	Do you like Django Reinhardt's music?	2006-09-13 18:45
2	Do you like Python?	2006-09-14 11:22

polls_choice

id	poll_id	choice	votes
1	1	Yes	0
2	1	No	0
3	1	Who is Django Reinhardt?	0
4	2	Yes	0
5	2	No	0

Many-to-Many

An author can have many books

A book can have many authors

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

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

id	name
1	Pro Django: Web Development Done Right

books_author

id	name
1	Adrian Holovaty
2	Jacob Kaplan-Moss

books_book_authors

id	book_id	author_id
1	1	1
2	1	2

Access the Relation

```
python manage.py shell

>>> 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

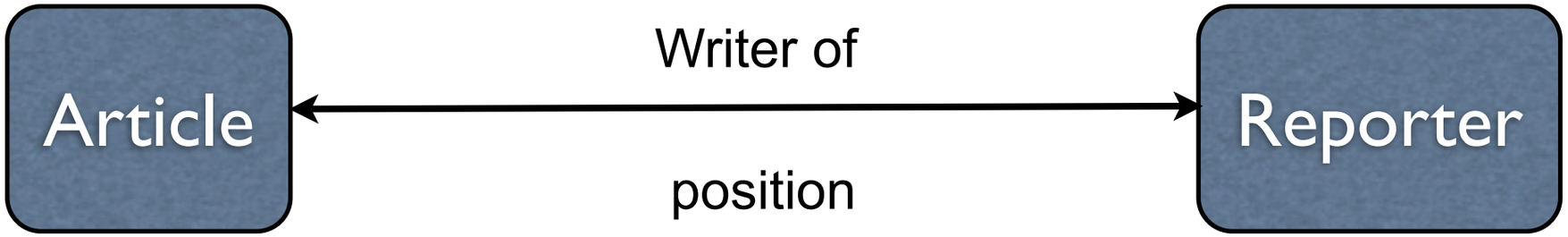
```
class Book(models.Model):
    name = models.CharField(maxlength=50)
    authors = models.ManyToManyField(Author, related_name='books')
```

```
>>> 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



Many-to-Many with Extra Data

```
from django.db import models

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

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

class Writer(models.Model):
    reporter = models.ForeignKey(Reporter)
    article = models.ForeignKey(Article)
    position = models.CharField(maxlength=100)
```

One-to-One

"The semantics of one-to-one relationships will be changing soon, so we don't recommend you use them"

Inheritance

```
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

http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx

Relating to self

```
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
```

save/delete hooks

```
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"
```

Choices

```
from django.db import models

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

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

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

get display

```
>>> 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'
```

And & or in Requests

```
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'))
```