Model

```
from django.db import models
from django.core.validators import (
    MaxValueValidator,
    MinValueValidator,
)

class Author(models.Model):
    # A very simple model example
    name = models.CharField(max_length=100)

class Book(models.Model):
    # Use a "ForeignKey" for a ManyToOne relationship
    author = models.ForeignKey(
        Author,
        on_delete=models.CASCADE,
    )

    title = models.CharField(max_length=100)
    release_date = models.DateField()

    # Very useful fields: "created" stores when it
    # originally was created, "last_updated" stores
    # whenever edited
    created = models.DateTimeField(
        auto_now_add=True)
    last_updated = models.DateTimeField(
        auto_now=True)

    # Multiple-choice fields, pattern is:
    # "internal code, human-readable label"
    CATEGORIES = (
        ("fict", "Fiction"),
        ("nonfict", "Non-fiction"),
    )
    category = models.CharField(
        max_length=10,
        default="fict",
        choices=CATEGORIES,
    )

    # example with custom validators
    num_stars = models.IntegerField(
        validators=[MaxValueValidator(5),
                    MinValueValidator(1)],
    )

    def __str__(self):      # Define __str__ to give
        return self.title   # string description


# ManyToMany relationships
class ReadingList(models.Model):
    books = models.ManyToManyField(Book)
```

Key words

id Automatically incrementing int, included with all Models

queryset Django terminology for the list-like data returned from database. Can be filtered further, or looped over.

CRUD Create, Read, Update, Delete - The four main operations of web application development.

Migration work-flow

```
python manage.py makemigrations # Detect changes
python manage.py showmigrations # Check status
python manage.py migrate # Apply any new migrations
```

DB Relationships

One-to-many An instance can be associated with an arbitrary number of other instances. *Example:* `Artist` can have released multiple `Albums`.

Many-to-many Freely associate any number to any number. *Example:* `Album` can have many `Tags`, and each Tag can have many albums. For Twitter, `User` can follow an arbitrary number of other `Users`.

One-to-one For every insteance of one model, there exists exactly one instance of another, effectively "splitting a model into two". *Example:* `Album` with `AlbumArtwork`.

Django CRUD Examples

ORM CRUD operations goes into `views.py`, to code the "business logic". The variables from the READ examples should be included in your templates to allow display to user.

```
##### CREATE
# Create a new book and save immediately in DB:
Book.objects.create(
    title="Great Expectations", num_stars=4)
# Alternate style: Create book, put into variable...
book = Book(title="Great Expectations")
book.num_stars = 4
book.save() # ...but only save to DB with .save()

##### READ
# Singular: Get 1 (and only 1) book that matches
book = Book.objects.get(title="Great Expectations")
print(book.author) # Template: {{ book.author }}
# Plural: Get all book(s) that match criteria
f_books = Book.objects.filter(category="fict")
for b in f_books: # Template: {% for b in f_books %}
    print(b.author) # Template: {{ b.author }}
# More complicated plural: All 4+ star, newest first
new_good_books = (Book.objects    # (parenthesis are
    .filter(num_stars__gt=3)      # for multi-lines)
    .order_by("-date"))

##### UPDATE
book = Book.objects.get(title="Great Expectations")
book.num_stars = 5  # Updates a single property
book.save()         # Saves the change to the DB
# Example of .update() on queryset (plural example):
nonfict = Book.objects.filter(category="nonfict")
nonfict.update(num_stars=5)  # Updates all books
# Example creating a many-to-many association:
rl = ReadingList.objects.get(title="Must read")
rl.books.add(book) # No need for ".save()" after add

##### DELETE
book = Book.objects.get(title="Great Expectations")
book.delete() # .delete() works on a single object...
Book.objects.filter(num_stars=1).delete() # or many
```