Chapter 3

MTV way
Configuring the database
Defining Models in Python
Installing the Model
Basic Data Access
Selecting and Deleting Objects
Changing Database Schema

The “Dumb” Way to Do Database Queries in Views

```
from django.shortcuts import render_to_response
import MySQLdb
def book_list(request):
    db = MySQLdb.connect(user='me', db='mydb',

    passwd='secret', host='localhost')
    cursor = db.cursor()
    cursor.execute('SELECT name FROM books
ORDER BY name')
    names = [row[0] for row in cursor.fetchall()]
    db.close()
    return render_to_response('book_list.html',
{'names': names})
```

The Django Way

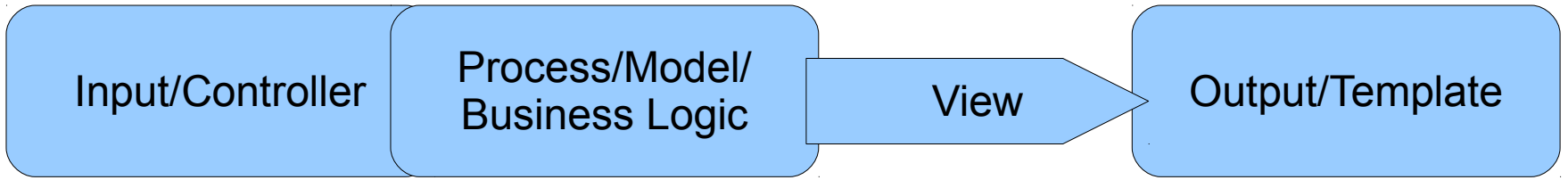
```
from django.shortcuts import render_to_response
from mysite.books.models import Book
def book_list(request):
    books = Book.objects.order_by('name')
    return render_to_response('book_list.html',
{'books':

books})
```

The MTV Development Pattern

- The MVC:

- The MTV



Settings.py

- DATABASE_ENGINE = ''
- DATABASE_NAME = ''
- DATABASE_USER = ''
- DATABASE_PASSWORD = ''
- DATABASE_HOST = ''
- DATABASE_PORT = ''

Checking the settings

- python manage.py shell
- from django.db import connection
- cursor = connection.cursor()
- If not configured properly, it will throw an error

Typical Errors

- You haven’t set the
- DATABASE_ENGINE setting yet.
- Environment variable
- DJANGO_SETTINGS_MODULE
- is undefined.
- Error loading _____ module: No
- module named _____.
- _____ isn’t an available database
- backend.
- database _____ does not exist
- role _____ does not exist

Your First App

- Difference between application and project
- A project is an instance of a certain set of Django apps, plus the configuration for those apps. Technically, the only requirement of a project is that it supplies a settings file, which defines the database connection information, the list of installed apps, the TEMPLATE_DIRS, and so forth.
- An app is a portable set of Django functionality, usually including models and views, that lives together in a single Python package.
- python manage.py startapp books

Defining Models in Python

- Why define model in Python?
- Introspection requires overhead and is imperfect.
- keeping everything in Python limits the number of times your brain has to do a “context switch.”
- Having data models stored as code rather than in your database makes it easier to keep your models under version control. This way, you can easily keep track of changes to your data layouts.
- SQL allows for only a certain level of metadata about a data layout. Most database systems do not provide a specialized data type for representing email addresses or URL.
- SQL is inconsistent across database platforms.

Creating the model

- Models.py
- Edit the settings.py
- INSTALLED_APPS → 'mysite.books',
- python manage.py validate
- python manage.py sqlall books
- Have Fun...
- But this does not touch actual database
- For that...
- python manage.py syncdb

Points to Ponder

- Table names are automatically generated by combining the name of the app (books) and the lowercased name of the model (publisher, book, and author). You can override this behavior
- As we mentioned earlier, Django adds a primary key for each table automatically
- By convention, Django appends "_id" to the foreign key field name
- The foreign key relationship is made explicit by a REFERENCES statement
- These CREATE TABLE statements are tailored to the database you're using, so database-specific field types such as auto_increment (MySQL), serial (PostgreSQL) are handled automatically

Some data

- >>> from books.models import Publisher
- >>> p1 = Publisher(name='Addison-Wesley',
- address='75 Arlington St.',
- ...city='Boston', state_province='MA',
  country='U.S.A.',
- ...website='http://www.addison-wesley.com/')
- >>> p1.save()
- >>> p2 = Publisher(name="O'Reilly", address='10
  Fawcett St.',
- ...city='Cambridge', state_province='MA',
  country='U.S.A.',
- ...website='http://www.oreilly.com/')
- >>> p2.save()
- >>> publisher_list = Publisher.objects.all()
- >>>publisher_list

Accomplishment

- To create an object, just import the appropriate model class and instantiate it by passing in values for each field.
- To save the object to the database, call the save() method on the object. Behind the scenes, Django executes an SQL INSERT statement here.
- To retrieve objects from the database, use the attribute Publisher.objects. Fetch a list of all Publisher objects in the database with the statement Publisher.objects.all().
-

Updating data

- >>> p = Publisher(name='Apress',
- ...address='2855 Telegraph Ave.',
- ...City='Berkeley',
- ...state_province='CA',
- ...country='U.S.A.',
- ...website='http://www.apress.com/')
- >>> p.save()
- >>> p.name = 'Apress Publishing'

Filtering Data

- >>> Publisher.objects.filter(name="Apress Publishing")
- >>> Publisher.objects.filter(country="U.S.A.", state_province="CA")
- >>> Publisher.objects.filter(name__contains="press")
- Gets translated to '%press%'
- Publisher.objects.get(name="Apress Publishing")
- Instead of a list (rather, QuerySet), only a single object is returned. Because of that, a query resulting in multiple objects will cause an exception
- >>> Publisher.objects.get(country="U.S.A.")

Ordering Data

- >>> Publisher.objects.order_by("name")
- >>> Publisher.objects.order_by("address")
- >>> Publisher.objects.order_by("country", "address")
- Most of the time you'll have a particular field you usually want to order by.
- In class definition create a subclass Meta
- class Meta:
- ordering = ["name"]
- Can achieve much more

Deleting Objects

- >>> apress = Publisher.objects.get(name="Addison-Wesley")
- >>> apress.delete()
- >>> Publisher.objects.all()
- [<Publisher: Apress Publishing>, <Publisher: O'Reilly>]
- Bulk deletion possible
- >>> publishers = Publisher.objects.all()
- >>> publishers.delete()
- >>> Publisher.objects.all()

Modifying Database schema

- Requires dbms shell
- Requires sql knowledge
- Alter/Drop etc...
- Requires server restart
- Please consult online document