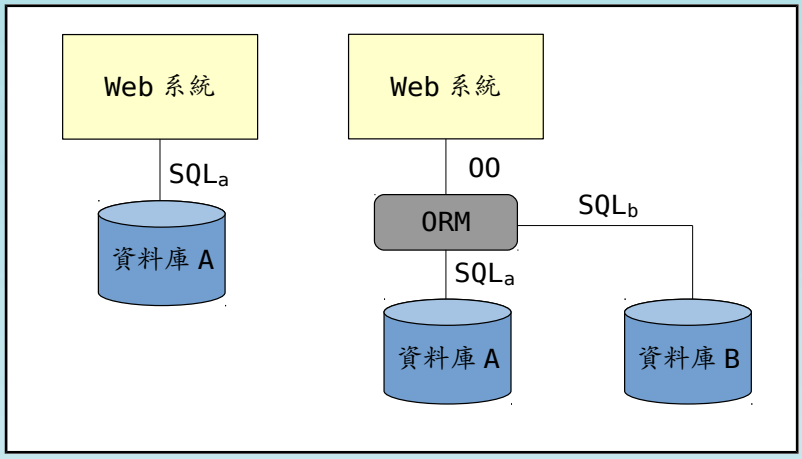
第7章 資料模型與資料庫

- Django 的資料庫模組(Database module)：物件關聯對應(Object-relational mapping, ORM)
  * ORM 是一個程式技術，以物件導向程式語言在不相容的資料系統之間做資料轉換
  * 再也不需要結構化查詢語言(Structural Query Language, SQL)
  * 可方便替換資料庫，不需要更改 Web 系統的程式



- Django 的資料模型(Data model)
  * Django model 是一個 Python 物件，用來描述資料模型
  * 我們可以直接利用 Python 物件來操作資料模型，而不需要透過 SQL

(1) 資料庫正規化

- 關聯式資料庫(Relational database)
  * 由一群相互關聯的資料表所組成
  * 表與表之間的關聯透過外來鍵來連繫

- 鍵值欄位
  * 主要鍵(Primary key)欄位
    # 依據該欄位可以找到唯一的一筆記錄
    # 主要鍵可能是一個或多個欄位的組合
  * 外來鍵(Foreign key)欄位
    # 該欄位對應到另一資料表的主要鍵

- 資料庫正規化(Database normalization)
  * 將資料表及欄位重組，以減少資料冗餘(Redundancy)

- 未正規化資料表(Unnormalized table，以訂單爲例)：

Order：

record#	orderNum	orderDate	productNum	productName	productDesc	numOrdered
1	40311	2015-10-10	304 633 684	手提電腦 電視機 洗衣機	ASUS 13 吋螢幕 LG 液晶螢幕 全自動洗衣脫水	7 1 4
2	40312	2015-10-10	128 304	個人電腦 手提電腦	Acer 17 吋螢幕 ASUS 13 吋螢幕	6 3
3	40313	2015-10-15	304	手提電腦	ASUS 13 吋螢幕	5

* 主要鍵爲 orderNum (畫底線)
* 未正規化資料表的問題：資料庫中每個欄位僅能儲存一個值，不能有重複群組(Repeating group)，例如 productNum、productName、productDesc、及 numOrdered

- 第一正規化(First normal form, 1NF)
  * 規格：沒有重複群組(亦即每個欄位只有一個值)
  * 轉換爲 1NF：擴展主要鍵讓每個重複群組資料都有主要鍵，其餘欄位則重複資料

  Order：

record#	orderNum	orderDate	productNum	productName	productDesc	numOrdered
1	40311	2015-10-10	304	手提電腦	ASUS 13 吋螢幕	7
2	40311	2015-10-10	633	電視機	LG 液晶螢幕	1
3	40311	2015-10-10	684	洗衣機	全自動洗衣脫水	4
4	40312	2015-10-10	128	個人電腦	Acer 17 吋螢幕	6
5	40312	2015-10-10	304	手提電腦	ASUS 13 吋螢幕	3
6	40313	2015-10-15	304	手提電腦	ASUS 13 吋螢幕	5

  * 上例中，轉爲 1NF 後，主要鍵變爲(orderNum, productNum)
  * 1NF 的問題：
    # 需要重複更新：例如要將「手提電腦」改爲「筆記電腦」，需要修改許多次
    # 資料不一致：多次更改資料，可能發生更改結果不相同的問題
    # 刪除一筆資料時，可能連同必要資料一併被刪除：例如刪除訂單 40311，連電視機產品資料都被刪除
    # 新增一筆資料需要許多空白資料：例如要增加一個新產品「智慧型手機」，僅能填入 productNum、productName、及 productDesc，其餘都是空白

record#	orderNum	orderDate	productNum	productName	productDesc	numOrdered
7			305	智慧型手機	HTC One	

- 第二正規化(Second normal form, 2NF)
  * 規格：符合第一正規化規格，且所有非主要鍵欄位必須功能相依(Functional dependent)於整個主要鍵
  * 功能相依(Functional dependency)：一個欄位的值依賴另一個欄位來決定，例如訂單日期是由訂單編號來決定(訂單日期無法決定訂單編號)

→ 如果主要鍵是單一欄位，1NF 資料表就符合 2NF 規格

* 轉換爲 2NF：

1. 依據所有主要鍵的組合來產生新的資料表設計：一次 1 個欄位，一次 2 個欄位，一次 3 個欄位，以此類推

   # 主要鍵欄位組合：orderNum, productNum, (orderNum, productNum)

2. 將其他欄位置於依賴最少主要鍵的資料表

   # orderNum: orderDate

   # productNum: productName, productDesc

   # (orderNum, productNum): numOrdered (外來鍵：orderNum 及 productNum，以星號表示)

Record:

record#	orderNum	orderDate
1	40311	2015-10-10
2	40312	2015-10-10
3	40313	2015-10-15

Product:

record#	productNum	productName	productDesc
1	304	手提電腦	ASUS 13 吋螢幕
2	633	電視機	LG 液晶螢幕
3	684	洗衣機	全自動洗衣脫水
4	128	個人電腦	Acer 17 吋螢幕

OrderItem (星號表「外來鍵」):

record#	orderNum*	productNum*	numOrdered
1	40311	304	7
2	40311	633	1
3	40311	684	4
4	40312	128	6
5	40312	304	3
6	40313	304	5

3. 將僅有主要鍵欄位的資料表刪除

* 2NF 的問題：

並未完全解決重複修改的問題：例如以下資料表符合 2NF 規格 (只要主要鍵是單一欄位，就符合 2NF)

Customer:

record#	customerNum	customerName	address	salesNum	salesName
1	108	張三	吉峰東路 168 號	41	約翰
2	233	李四	吉峰東路 169 號	22	湯姆
3	254	王五	吉峰東路 170 號	38	瑪莉
4	431	趙六	吉峰東路 171 號	74	東尼
5	779	陳七	吉峰東路 172 號	38	瑪莉
6	800	沈八	吉峰東路 173 號	74	東尼

更改銷售員的名字還是牽涉重複修改

重複修改會產生資料部一致
刪除一個銷售員導致必要資料也被刪除
增加一個新的銷售員會讓許多欄位爲空白

- 第三正規化(Third normal form, 3NF)
  * 規格：符合 2NF 規格，且沒有非主要鍵欄位彼此功能相依
  * 一般 Web 系統的資料模型符合 3NF 就足夠了，其他正規化：BCNF, 4NF, 5NF, ...
  * 轉換爲 3NF：將所有相依於非主要鍵的欄位移至另一資料表，並以被相依的欄位爲主要鍵
    → 上例中，salesName 相依於 salesNum，將 salesName 欄位移走

Customer:

record#	customerNum	customerName	address	salesNum*
1	108	張三	吉峰東路 168 號	41
2	233	李四	吉峰東路 169 號	22
3	254	王五	吉峰東路 170 號	38
4	431	趙六	吉峰東路 171 號	74
5	779	陳七	吉峰東路 172 號	38
6	800	沈八	吉峰東路 173 號	74

Sales:

record#	salesNum	salesName
1	41	約翰
2	22	湯姆
3	38	瑪莉
4	74	東尼

(2) 建立系統管理者

- 每個 Django project 都需要超級使用者(Superuser)
  * Django 提供功能強大的管理者頁面，用來管理資料
  * Django 內建有 User model，用來儲存使用者資料，可在裡面建立 Superuser 帳號：

```
$ source <venv>/bin/activate
(<venv>)$ cd <projRoot>
(<venv>)$ python manage.py createsuperuser
Username (leave blank to use '<username>'): admin
Email address: admin@gmail.com
Password:  admin12345
Password (again): admin12345
Superuser created successfully.
```

註：亦可使用其他帳號名稱，但應使用較複雜的密碼，電子信箱眞實性不重要

* 可以建立許多超級使用者
# 進入 admin 頁面：localhost:8000/admin/
# 嘗試加入一個新使用者

(3) 建立模型

- 規劃 article 的 Data model 的表格：Article 與 Comment

Article:

title (Char)	content (Text)	likes (Integer)
-	-	-

Comment:

article (ForeignKey)	content (Char)
-	-

* Django 使用 Python 的 class (類別)來建立 model (Model 名稱通常大寫)
* 編輯 article/models.py，加入以下內容：

```
from django.db import models


class Article(models.Model):
    title = models.CharField(max_length=128, unique=True)
    content = models.TextField()
    likes = models.IntegerField(default=0)

    def __str__(self):
        return self.title


class Comment(models.Model):
    article = models.ForeignKey(Article)
    content = models.CharField(max_length=128)

    def __str__(self):
        return self.article.title + '-' + str(self.id)
```

from ...：從 django.db 匯入 models 套件
class Article ...：宣告一個 Article model (是一個 Python class)，繼承 models.Model
- 有 3 個欄位：
  * title：文章標題，字元欄位格式(單行)，最多 128 個字元，標題爲唯一
  * content：文章內容，文字欄位格式(大量、包含 Enter)
  * likes：按讚次數，整數欄位格式，預設值爲 0
- def __str__(self)...：定義一個__str__() 方法，回覆 self.title 值，這是在範本中此物件預設顯示的值，在 admin 介面亦顯示此值

class Comment(...)：宣告一個 Comment model，繼承 models.Model

- 有 2 個欄位：

  * article：外來鍵(Foreign key)，指向 Article model，指定留言所屬的文章

    # 一篇文章可以有許多留言，一個留言只能針對一篇文章，因此是多對一的關係，利用外來鍵關聯

    # 外來鍵的欄位名稱習慣上就設定爲所指向的 model 名稱，但改爲小寫

  * content：留言內容，字元欄位格式(單行)，最多 128 個字元

- def __str__(self)：定義一個__str__() 方法，回覆值爲 self.article.title + '-' + str(self.id) (亦即將文章標題串上該物件的 id)

  * Django model 中，欄位資料的存取亦使用點號方法，因此從留言(Comment)連到其所屬文章(Article)，再連到該文章的標題(title)，就可使用一連串的點號連結：coment.article.title

註：欄位名稱不可使用 Django model API 的名稱，例如 clean, save, delete 等

- Django model 提供許多欄位型態，常用的如下：

* CharField：字元(單行)	* ImageField：影像
* DateField：日期	* IntegerField：整數
* DateTimeField：日期時間	* TextField：文字(多行、大量)
* EmailField：電郵	* ForeignKey：多對一關聯
* FileField：檔案	* OneToOneField：一對一關聯
* FloatField：浮點數	* ManyToManyField：多對多關聯

(4) 資料庫遷移

- 執行 makemigrations

  * 右鍵點專案 → Django → Custom Command → Command: makemigrations → OK

```
Migrations for 'article':
  article/migrations/0001_initial.py:
    - Create model Article
    - Create model Comment
Finished "<...>/webapps/git/blog/blog/manage.py makemigrations" execution.
```

  * makemigrations 在 article/migrations/ 目錄下建立 0001_initial.py 程式檔用來產生資料表，內容如下：

```
# -*- coding: utf-8 -*-
# Generated by Django 1.10.2 on 2016-10-08 13:34
from __future__ import unicode_literals

from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):

    initial = True

    dependencies = [
```

```
    ]

    operations = [
        migrations.CreateModel(
            name='Article',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('title', models.CharField(max_length=128, unique=True)),
                ('content', models.TextField()),
                ('likes', models.IntegerField(default=0)),
            ],
        ),
        migrations.CreateModel(
            name='Comment',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('content', models.CharField(max_length=128)),
                ('article', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='article.Article')),
            ],
        ),
    ]
```

如果使用者並未定義 id 欄位，Django 會自動加了一個 id 欄位(正整數)
→ 最好不要自行定義 id 欄位，交由 Django 來設定及操作較好

可利用以下指令了解 0001_initial.py 所產生的 SQL 指令(格式：python manage.py sqlmigrate <app> <migrateNumber>)，例如：

python manage.py sqlmigrate article 0001

- 執行 Migrate

* 右鍵點專案 → Django → Migrate

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, article, sessions
Running migrations:
  Applying article.0001_initial... OK
Finished "/home/yltang/webapps/git/blog/blog/manage.py migrate" execution.
```

- 在 article/admin.py 登記 Article 及 Comment，即可在 admin 頁面檢視資料

```
from django.contrib import admin
from article.models import Article, Comment


admin.site.register(Article)
admin.site.register(Comment)
```

* 進入 admin 網頁：localhost:8000/admin/並輸入帳號密碼

Django 會在各個 Model 的名稱後面加上 s 成爲複數
可嘗試新增 Article 與 Comment

(5) Django ORM

- ORM (Object-relational mapping，物件關聯對應)
  * ORM 是一個工具，讓使用者能以物件導向的方式操作資料庫，不再使用 SQL 語言
  * 常用的資料庫操作：新增、讀取、修改、刪除、及查詢(「增讀改刪查」：Create, read, update, delete, search, CRUDS)，以 Article 爲例

```
# 新增(Create)
    直接新增並儲存物件
    Article.objects.create(...)
    Article.objects.get_or_create(...)

    或產生物件，設定各欄位值，再儲存
    article = Article()
    article.title = ...
    article.content = ...
    article.save()

# 讀取(Read)
    Article.objects.get(...)                    # 取出一筆符合條件的資料

# 修改(Update)
    取出物件，修改欄位值，再儲存
    article = Article.objects.get(...)
    article.title = ...
    article.content = ...
    article.save()

# 刪除(Delete)
    取出物件，刪除
    article = Article.objects.get(...)
    article.delete()

# 查詢(Search)
    Article.objects.all()                       # 取出所有資料
```

```
Article.objects.get(...)                    # 取出一筆符合條件的資料
Article.objects.filter(...)                 # 取出多筆符合條件的資料
Article.objects.exclude(...)                # 取出多筆不符合條件的資料
Article.objects.order_by(...)               # 取出所有資料並排序
Article.objects.filter(...).order_by(...)   # 取出多筆符合條件的資料並排序
```

(6) 建立資料填充程式

- 程式開發期間或系統第一次部署時，將資料一筆一筆輸入資料庫太耗時，通常會撰寫自動填充資料的程式，此種程式稱爲資料填充程式(Data population script)
  - * 所需填充的資料：
    - # 測試資料：程式開發或測試階段所需要的測試資料
    - # 基本資料：程式第一次部署時必須要填入的資料(例如：admin 帳號、產品基本資料、或系統設定資料)
  - * 將所有填充程式集中在 populate 目錄裡：在專案目錄下新增 populate 目錄
  - * 新增 populate/__init__.py 檔案：將 populate 目錄設定爲 Python package
  - * 新增 populate/base.py：此爲各個填充程式所需之設定檔，由於填充程式是直接執行，而非透過伺服器，故需先設定 Django 環境

base.py:

```
import os
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'blog.settings')
import django
django.setup()
```

  - * 新增部落格文章填充程式

populate/article.py:

```
from populate import base
from article.models import Article, Comment
import random


def populate():
    print('Populating Article and Comment...', end='')
    titles = ['如何像電腦科學家一樣思考', '10 分鐘內學好 Python', '簡單學習 Django']
    comments = ['文章眞棒', '並不認同您的觀點', '借分享', '學好一個程式語言或框架並不容易']
    Article.objects.all().delete()
    Comment.objects.all().delete()
    for title in titles:
        article = Article()
        article.title = title
        for j in range(20):
            article.content += title + '\n'
        article.likes = random.randint(0, 10)
```

```
        article.save()
        for comment in comments:
            Comment.objects.create(article=article, content=comment)
    print('done')


if __name__ == '__main__':
    populate()
```

首先匯入所需資源，注意，import base 一定要放在第一行：需先設定 Django 後，其餘程式才能正常運作

def populate()：新增資料的函式

- titles = ... , comments = ... 預設文章標題及留言內容
- Article.objects.all().delete(), Comment.objects.all().delete()：刪除所有文章及留言資料
- for title in titles：使用 titles 資料當作標題
- article = Article()：新增一筆 Article instance
- article.title = title：設定文章標題
- for j in range(...):
  article.content = ...：設定留言內容爲多行文字(使用標題當內容)
- article.likes = random.randint(...)：設定隨機按讚次數
- article.save()：存入資料庫
- Comment.objects.create(article=article, content=comment)：新增並儲存此文章所屬留言

# if __name__ == '__main__':
  populate()

→ 如果此模組是直接執行而不是被匯入，就呼叫 populate() 函式來新增資料

- Python 模組的執行與匯入：
  * 模組被直接執行時，Python 會將內建變數__name__內容設定爲__main__
  * 模組被匯入時，Python 會__name__設定爲該模組名稱，例如：

```
import math
print(math.__name__)
```

  * 因此判斷__name__變數，就可知道是直接執行或是被匯入

* 執行填充程式：啓動虛擬環境，移到專案目錄，然後執行模組

```
$ source <venv>/bin/activate
(<venv>)$ cd <projRoot>
$ python -m populate.article
Populating Article and Comment...done
```

* 在管理者頁面檢視 Article 資料表：

- 也可以利用填充程式建立 admin 帳號

populate/admin.py:

```
from populate import base
from django.contrib.auth.models import User


def populate():
    print('Creating admin account...', end='')
    User.objects.all().delete()
    User.objects.create_superuser(username='admin', password='admin', email=None)
    print('done')


if __name__ == '__main__':
    populate()
```

* 首先匯入相關模組，User 是 Django 的內建模型，用來儲存使用者
* def populate() ：新增 admin 帳號的函式
  # User.objects.all().delete() ：清除所有 User model 裡的資料
  # User.objects.create_superuser ：利用 create_superuser() 函式新增 admin 帳號
* 執行填充程式：

```
$ python -m populate.admin
Creating admin account...done
```

- 整合所有填充程式：一次執行完所有填充程式

* 新增整合填充程式

populate/all.py:

```
from populate import admin, article


admin.populate()
article.populate()
```

匯入所需模組

執行 admin 與 article 模組
由於 all.py 模組一定是直接執行，不會被匯入，因此不需要判斷是被執行或匯入(if __name__ == '__main__')

* 執行整合填充程式：一次將所有填充程式執行完畢

```
$ python -m populate.all
Creating admin account...done
Populating Article and Comment...done
```

* 至管理者頁面檢視 Article 及 Comment 模型，確認內容正確
  → 資料顯示的值即爲該 Model 裡__str__() 方法所回覆的值

(7) 客製化 admin 頁面

- admin 頁面裡資料的顯示模式可以做許多客製化，例如：

* 客製化 Comment model：在 article/admin.py 加入

```
from django.contrib import admin
from article.models import Article, Comment


class CommentModelAdmin(admin.ModelAdmin):
    list_display = ['article', 'content']

    class Meta:
        model = Comment


admin.site.register(Article)
admin.site.register(Comment, CommentModelAdmin)
```

增加一個 CommentModeAdmin 類別(繼承 admin.ModelAdmin)
客製化頁面顯示清單(list_diaplay)：顯示 article 與 content 欄位
class Meta 是一個類別容器(Class container)，內含有關該類別的詮釋資料(稱爲 Metadata)，例如：排序、權限、所使用的 Model 等
在 admin.site.register() 裡新增繼承 CommentModeAdmin 類別
→ 測試：清單顯示 2 個欄位(ARTICLE, CONTENT)

* 增加連結(list_diaplay_links)：透過 article 來連結(此項爲預設)

```
class CommentModelAdmin(admin.ModelAdmin):
    list_display = ['article', 'content']
    list_display_links = ['article']
```

* 增加過濾器(list_filter)：設定右方過濾欄位爲 article 與 content

```
class CommentModelAdmin(admin.ModelAdmin):
    ...
    list_display_links = ['article']
    list_filter = ['article', 'content']
```

→ 測試：點選文章標題就會列出所屬留言

* 增加搜尋欄位(search_fields)：設定搜尋欄位爲 content

```
class CommentModelAdmin(admin.ModelAdmin):
    ...
    list_filter = ['article', 'content']
    search_fields = ['content']
```

→ 測試：出現搜尋欄位

* 增加編輯欄位(list_editable)：設定 content 欄位可編輯

```
...

class CommentModelAdmin(admin.ModelAdmin):
    ...
    search_fields = ['content']
    list_editable = ['content']
```

→ 測試：可編輯內容

* 其他客製化：https://docs.djangoproject.com/es/1.9/ref/contrib/admin/#modeladmin-options

(8) 增加 Model 欄位

- 增加 Article model 欄位

* 管理者發表文章時，應記錄發表時間，因此新增欄位 pubDateTime，此外，規劃文章依照發表時間順序顯示：

```
class Article(models.Model):
    title = models.CharField(max_length=128, unique=True)
    content = models.TextField()
    pubDateTime = models.DateTimeField(auto_now_add=True)
    likes = models.IntegerField(default=0)

    def __str__(self):
        return self.title

    class Meta:
        ordering = ['-pubDateTime']
```

新增 pubDateTime 欄位(發表日期時間)

- DateTimeField：日期時間格式
- auto_now_add=True：在新增物件時自動設定爲當時時間，建立之後即無法修改
- 還有另一種自動設定時間的方式 auto_now=True，例如：

```
updated = models.DateTimeField(auto_now_add=False, auto_now=True)
```

→ 第一次儲存物件時不要設定此欄位資料，但之後每次儲存時都設定新時間

class Meta：在詮釋資料中設定模型的特性

- ordering = ['-pubDateTime'])：物件存入資料庫時依照時間由近至遠排序(使用

串列儲存)，可設定多個欄位排序

* 註：如果在 Model 沒有設定排序，就必須在 views 中排序，例如：

```
articles = Article.objects.order_by('-pubDateTime')
```

Web 系統設計理念：

Fat models, thin views, and stupid templates.

- 有關資料儲存的規格最好在 models (或 forms)裡執行，所以程式很多(Fat)
- Views 程式用來處理瀏覽器的請求，速度要很快，所以應該輕薄短小(Thin)
- 範本標籤的邏輯應該越單純越好(Stupid)

- 增加 Comment model 欄位

* 使用者留言時，也應記錄留言時間，因此新增欄位 pubDateTime

```
class Comment(models.Model):
    article = models.ForeignKey(Article)
    content = models.CharField(max_length=128)
    pubDateTime = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.article.title + '-' + str(self.id)
```

- 執行資料庫遷移：

* 執行 Makemigrations：在 article/migrations/ 目錄下產生 0002*.py 檔案

```
You are trying to add the field 'pubDateTime' with 'auto_now_add=True' to article without a default; the
  database needs something to populate existing rows.
 1) Provide a one-off default now (will be set on all existing rows)
 2) Quit, and let me add a default in models.py
Select an option: 1
Please enter the default value now, as valid Python
You can accept the default 'timezone.now' by pressing 'Enter' or you can provide another value.
The datetime and django.utils.timezone modules are available, so you can do e.g. timezone.now
Type 'exit' to exit this prompt
[default: timezone.now] >>> <Enter>
```

- <Enter>表示接受輸入建議：timezone.now (此程序會出現 2 次，因爲有 2 個欄位需給初始值)

```
Migrations for 'article':
  article/migrations/0002_auto_20161010_1134.py:
    - Change Meta options on article
    - Add field pubDateTime to article
    - Add field pubDateTime to comment
Finished "<...>/webapps/git/blog/blog/manage.py makemigrations" execution.
```

* 執行 Migrate：

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, article, sessions
Running migrations:
  Applying article.0002_auto_20161010_1134... OK
Finished "<...>/webapps/git/blog/blog/manage.py migrate" execution.
```

Django 在 article/migrations 目錄裡產生 0002_auto_....py 檔案，內容如下：

```
# -*- coding: utf-8 -*-
# Generated by Django 1.10.2 on 2016-10-10 03:34
from __future__ import unicode_literals

from django.db import migrations, models
import django.utils.timezone

class Migration(migrations.Migration):
    dependencies = [
        ('article', '0001_initial'),
    ]
    operations = [
        migrations.AlterModelOptions(
            name='article',
            options={'ordering': ('-pubDateTime',)},
        ),
        migrations.AddField(
            model_name='article',
            name='pubDateTime',
            field=models.DateTimeField(auto_now_add=True, default=django.utils.timezone.now),
            preserve_default=False,
        ),
        migrations.AddField(
            model_name='comment',
            name='pubDateTime',
            field=models.DateTimeField(auto_now_add=True, default=django.utils.timezone.now),
            preserve_default=False,
        ),
    ]
```

- dependencies：此程式依賴('article', '0001_initial')版本
- migrations.AlterModelOptions：更改選項
- migrations.AddField：增加欄位

* 重新執行文章填充程式(python -m populate.article)，並至 admin 頁面確認資料正確

(9) 重建資料庫

- 重建資料庫

* 系統開發期間，常因 Model 欄位頻繁的修改，造成資料庫錯亂，有時需要完全刪除資料庫，然後重建，步驟如下：

1. 停止伺服器
2. 刪除再重建資料庫

$ sudo -i -u postgres

[sudo] password for <username>: xxxxxxxx

postgres@<username>:~$ dropdb blogDB

```
postgres@<username>:~$ createdb blogDB
postgres@<username>:~$ psql
postgres=# grant all privileges on database "blogDB" to "blog";
GRANT
postgres=# \q
postgres=# exit
```

3. 刪除所有<app>/migrations/00*.py 檔案

   $ find . -type f -name 00*.py -exec rm {} \;

   - 也可利用 Nautilus 的搜尋功能
4. 資料庫遷移：makemigrations 與 migrate
5. 填充資料：python -m populate.all

(10) Push to Github

- 在系統開發階段，各開發者有自己的資料庫，因此不需分享遷移檔案
  * 在 .gitignore 裡加入以下內容，將 00*.py 檔排除

```
*~
__pycache__
*.pyc
00*.py
```

  * 但 EGit 在 Commit 時，可能還是會挑到 .gitignore 裡列舉的檔名型態，需人工勾選

- Now push to Github

(11) 練習

- 練習 1：刪除資料庫，重建資料庫，並重新填入資料

- 練習 2：新增 book app，在其中新增 Book model
  * 包含以下欄位：書名、作者姓名、出版商、出版日期、印刷版次、及售價
  * 撰寫填充程式，建立至少 5 本書之資料