Introduction to Django and MariaDB

Jesús Cabo Culebras

Francisco Javier García Ruiz

Infrastructure: What have we chosen and why?

MariaDB

MariaDB is a open source database management system, that manage the data and the way to access to it. It is a fork of MySQL, created after the acquisition of Sun Microsystems by Oracle, and one of the main goals of this software is to be compatible with mysql, but including new features focused in performance. MariaDB is rapidly increasing its market share due it is totally free and open source and its compatibility with MySQL.

We chose MariaDB because is really easy to work with it if you have worked with MySQL and because probably MariaDB will take advantage over MySQL shortly.

Django

Django is a web framework that helps to create webpages easily. It is found between database layer and web layer (HTML) and it makes a connection among them. To perform this communication it is used Python language.

Actually, a database is allocated in a server and to make a connection a php file is used, writing that file in php language. Using Django we realise a .py file where are developed the main functions that will be call from a HTML client file. With this strategy, the developer do not create a huge HTML file and it is not understandable very often because it could be written in different language as javascript, php and html.

Although Django helps you with the connection with the database, it comes with an object-relational mapper (transformation of data between an Object Oriented model and a Relational database) in which you describe your database layout in Python code.

Installation

We used clean installation of Debian 6 as OS, but the installation is similar in the rest of Linux systems.

Install MariaDB

To download MariaDB we have to go to the official webpage (link) and follow the instructions to add the repository and install mariadb-server. In our case we have to follow the next steps:

1. First, run the following command to install the signing key:
   sudo apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db

2. You now need to create a custom MariaDB sources.list file. To do so, copy and paste the following into a file under /etc/apt/sources.list.d/ (we suggest naming the file MariaDB.list or something similar), or add it to the bottom of your /etc/apt/sources.list file.
   # MariaDB 5.5 repository list file
   deb http://mirror.vpsfree.cz/mariadb/repo/5.5/debian squeeze main
   deb-src http://mirror.vpsfree.cz/mariadb/repo/5.5/debian squeeze main

3. Once the key is imported and the repository added you can install MariaDB with:
   sudo apt-get update
   sudo apt-get install mariadb-server

Configure MariaDB

To access to the MariaDB query evaluator, we have to use the command
mysql -u user -p

and, inside the database management system, create a database and an user for it
create database equipment;
use equipment;
create user 'userName' identified by 'password ;
grant all privileges on equipment. to userName;*

to create the structure we have created a SQL file. If the file is named createDB.sql we should use the next command:
mysql -u root -p < createDB.sql

Install Django

We have several ways to install Django over a Linux platform. The easiest way is to install it using the package management system of the operative system.

aptitude install python-django

But this method installs an old version (1.2.3), so we will use a manual way. First we have to install some requisites.

1. Install CURL to download files
   apt-get install curl
2. Install SETUPTOOLS. It is a requirement for PIP
   sudo apt-get install python-setuptools
3. Install PIP. PIP is a tool for installing and managing Python packages. We have to download PIP package with curl and execute it with python (assuming python is installed on the system)
   curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py
   [sudo] python get-pip.py
4. Install DJANGO. We will install Django using PIP. We will install the MySQL connector too(valid for MariaDB).
   pip install Django
   apt-get build-dep python-mysqldb
   pip install MySQL-python

Configuring Django

To create project, where we will define the some configuration parameters, we have to execute the following command. It will create the structure in the current directory, in our case we will name the project 'equipment'.

django-admin.py startproject <projectName>

The next step is to define some configuration parameters related with the connection with the database in the file *<projectName>/settings.py.* Change the following keys in the DATABASES 'default' item to match your database connection settings.

- ENGINE – Either 'django.db.backends.postgresql_psycopg2', 'django.db.backends.mysql', 'django.db.backends.sqlite3' or 'django.db.backends.oracle'. Other backends are also available.
- NAME – The name of your database. If you're using SQLite, the database will be a file on your computer; in that case, NAME should be the full absolute path,

including filename, of that file. If the file doesn't exist, it will automatically be created when you synchronize the database for the first time (see below). When specifying the path, always use forward slashes, even on Windows (e.g. C:/homes/user/mysite/sqlite3.db).

- USER – Your database username (not used for SQLite).
- PASSWORD – Your database password (not used for SQLite).
- HOST – The host your database is on. Leave this as an empty string (or possibly 127.0.0.1) if your database server is on the same physical machine (not used for SQLite).

In our case the result will be something like this:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'equipment',
        'USER': 'admin1',
        'PASSWORD': 'admin',
            'HOST': '',                  # Empty for localhost through domain sockets or '127.0.0.1' for localhost through TCP.
        'PORT': '',                  # Set to empty string for default.
    }
}
```

And we have to define the timezone and language parameters

```
# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
# In a Windows environment this must be set to your system time zone.
TIME_ZONE = 'Europe/Prague'

# Language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = 'en-gb'
```

After configure it we have to synchronize the Django with the database. Manage.py is in the project directory.

python manage.py syncdb

This command will ask us to create a user and password, we create a python-superuser named 'admin1' pass 'admin' for example.

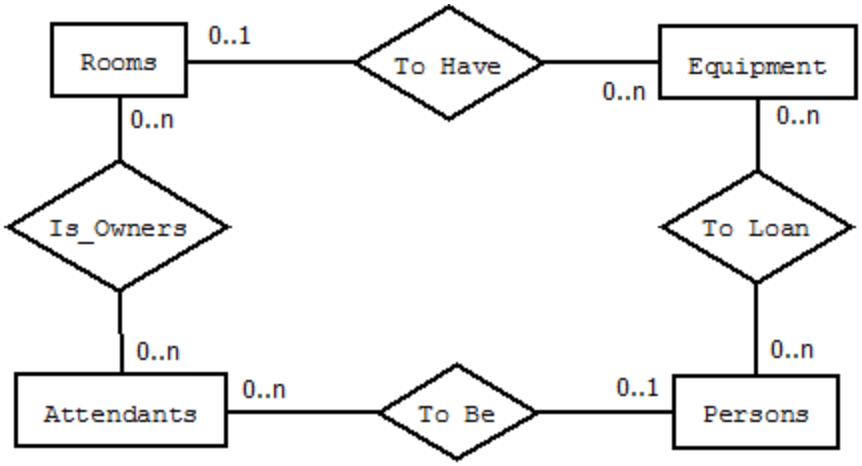
Importing a project

To import a project we have to copy our complete project to whichever directory and we can start working with it. We have to keep in mind that we must have all Django installed, as well as the database, and reconfigure the settings.py file with new values if is required.

Database

The picture below is representing the database design through E-R model that will be explained in the next section. Is an example about rooms at the university, the owners of the rooms (attendants) and the equipment that it is the rooms and it could be book by different persons.

In this model it is also possible seeing cardinality between different entities. This will be important at time to create different tables, or classes modules in Django, to work with this database.



Creating an application

Once we have the project correctly configured we can create an application. To do it we have to execute the command: *python manage.py startapp <appName>*

`python manage.py startapp equipmentApp`

The structure of the project and application and the files that we need to configure are the next:

- projectName/
    - settings.py : configure DB, timezone, etc.
    - urls.py : redirection of the requests
- applicationName/
    - models.py : definition of the model (data structure) and relation with the DB
    - urls.py : sends the requests to the corresponding view
    - views.py : application logic
    - templates/applicationName/
        - *.html : html templates

Database and Django Relation

In order to create the DB structure, and use the object-relational mapper included in Django, we have two ways:

1. Create the classes in the models.py file and autogenerate the SQL file

   Create the classes in models.py

   In settings.py, in the INSTALLED_APPS label add the name of the project

   `'equipmentApp'`

   and execute the following commands

   `python manage.py sql equipmentApp`

python manage.py syncdb

One simple example of a class can be

```
class TutorialRoom(models.Model):
room_id = models.CharField(max_length=100)
description = models.TextField()
```

and will be converted to

```
CREATE TABLE `tutorial_tutorialroom` (
  `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
  `room_id` varchar(100) NOT NULL,
  `description` longtext NOT NULL);
```

2. Create the structure in the database and autogenerate the models.py file

   python manage.py inspectdb > models.py

   Replace the new model.py in ./equipmentApp/models.py

   Clean up the information (ex. undesired classes) and resynchronize the database

   python manage.py syncdb

In the project we will use the second possibility because we already have the database done. We can try and play with the model with an interpreter.

python manage.py shell

```
>>> from equipmentApp.models import Room  # Import the model classes we just wrote.
# No rooms are in the system yet.
>>> Room.objects.all()
[ ]
# Create a new Room.
>>> r = Room(room_id="J345", description="Laboratory")
# Save the object into the database. You have to call save() explicitly.
>>> r.save()
```

Creating the views

The views are the implementation of the logic of the application. It receives a request and returns a response. A really simple example can be:

```
def index(request):
 message=”Hello, world!”
  return HttpResponse(message)
```

Connecting it with urls files

Using the urls files we define where have to go the requests.
In the project urls.py file we define to wich application have to go the request. So to make the application work we send the requests to the correct application’s urls file.

equipment/urls.py

```
urlpatterns = patterns('',
url(r'^equipment/', include('equipmentApp.urls'))
```

And in the application urls file we send the requests to the appropriate views function

equipmentApp/urls.py

```
 from django.conf.urls import patterns, url
 from equipmentApp import views
# url (regular expression, function, indicator)
urlpatterns = patterns('', url(r'^$', views.index, name='index') )
```

HTML templates

The templates are html files with inlaid python code that add functionality. We have to indicate in the file settings.py where are the templates located

```
TEMPLATE_DIRS = (
  '/home/admin1/equipment/equipmentApp/templates'
```

One example of a part of a template that lists the rooms is

```
<body>
    <h1 id="title">Rooms Registered</h1>
    {% if rooms %}
        <ul class="light-box">
        {% for room in rooms %}
            <li>{{room}}</li>
        {% endfor %}
```

```
        </ul>
    {% else %}
        <p>No rooms registered</p>
    {% endif %}
```

The object rooms is passed as context by the view. Lets see an example:

```
def index(request):
    rooms = Room.objects.all()
    context = {'rooms':rooms}
    return render(request,'equipmentApp/index.html', context)
```

We pass to the render the object context (key-value dictionary) with all the Room records in the database and are sent to the template.

Trying it

Django includes a web server for developing purposes.

```
python manage.py sql runserver
```

And then we can access through a web browser using the local url that will go to the index defined in the urls.py file

```
http://127.0.0.1:8000/equipmentApp/
```

Interesting links

Resources and more information

https://mariadb.org/

http://djangoproject.com

Installation information

https://downloads.mariadb.org/mariadb/repositories/

http://www.pip-installer.org/en/latest/installing.html#using-the-installer