

PYTHON

UNI – FIIS
Glen Rodríguez R.

Base de datos

- Es una abstracción encima del sistema de archivos del Sistema Operativo, que permite crear, almacenar, leer y modificar datos persistentes.
- Persistencia: el dato sigue disponible después que: el software se colgó, o el sistema operativo se apagó, o el programa (en Python) terminó.

Por qué se usan?

- En líneas generales, una aplicación almacena datos y los procesan / muestras en un formato o plantilla útil. Por ejemplo, Google Maps guarda datos sobre barrios, comercios y calles, y provee de información sobre las rutas para ir de un barrio a un centro comercial.
- La BD permite tener datos almacenados de forma estructurada, confiable y rápida. También proveen un marco lógico predefinido para razonar como grabar y leer datos.

Bases de datos relacionales

- Tablas
- Filas
- Columnas
- Claves primarias
- Relaciones y claves foráneas
- Joins

Usando bases de datos en Python

- Las interfaces entre Python y bases de datos deberían seguir el estándar Python DB-API. La mayoría lo hace.
- Bajo este estándar se han hecho módulos que enlazan Python con: mySQL, PostgreSQL, MS SQL Server, Informix, Oracle, Sybase, SQLite
- Otra opción es usar el conector ODBC
- La mayor parte NO vienen incluidos con Python. Hay que bajarlos por separado

Instalando los Módulos

- Un modulo se puede bajar desde linea de comandos usando el comando "pip". ay que tener conexión a Internet
- Formato
- `C:\> pip install <nombre del módulo>`
- Ejemplo
- Bajemos el modulo para acceder a MySQL
- `C:\>pip install pymysql`

Prerequisitos

- Hemos creado la base de datos "testdb".
- Hemos creado la tabla "empleados" en "testdb".
- Esa tabla tiene las columnas "nombres", "apellidos", "edad", "sexo" y "salario".
- El usuario "root" con password "password" tiene acceso a "testdb".
- Se ha instalado el pymysql correctamente.
- Sabemos lo básico de SQL y de MySQL.

Conectándose a la BD

```
import pymysql

# Abriendo conexión
db = pymysql.connect("localhost","root","password","testdb" )

# instanciando un objeto cursor
cursor = db.cursor()

# usando el método execute(), ejecutar un query SQL
cursor.execute("SELECT VERSION()")

# Leer una sola fila de los resultados del query
data = cursor.fetchone()

print ("Versión de la base de datos : %s " % data)

# disconexión
db.close()
```


Objeto cursor

- Así como un archivo de texto tiene su variable “manejador de archivo”, el resultado de una sentencia SQL también tiene su “manejador”, que és un objeto cursor.
- No es obligatorio usarlo (se puede hacer queries con la conexión), pero es recomendable si se hace más de una operación de BD en el programa

Creando la tabla

```
import pymysql

db = pymysql.connect("localhost","root","password","testdb" )
cursor = db.cursor()

# Eliminar la tabla si ya existia
# OJO solo se hace por que es un ejemplo en clase.
# Nunca hacerlo en la vida real
cursor.execute("DROP TABLE IF EXISTS empleados")

# Creando la tabla
sql = """CREATE TABLE empleados (
    nombres CHAR(20) NOT NULL,
    apellidos CHAR(20),
    edad INT,
    sexo CHAR(1),
    salario FLOAT )"""

cursor.execute(sql)
db.close()
```

Excepciones

- Los “execute” pueden dar errores, por eso debería ponerse dentro de un try:

try:

...

 cursor.execute(sql)

 # ...

except Exception as e:

 print (e)

finally:

 connection.close()

- Tipos de errores: AttributeError, pymysql.Error
(pymysql.err.ProgrammingError, pymysql.err.DataError,
pymysql.err.IntegrityError, pymysql.err.NotSupportedError,
pymysql.err.OperationalError)

Insertando registros

```
import pymysql
db = pymysql.connect("localhost","root","password","testdb" )
cursor = db.cursor()

vNombre=input("Entre nombres de empleado: ")
vApellido=input("Entre apellidos de empleado: ")
vEdad=int(input("Entre edad de empleado: "))
vSexo=input("Entre sexo (F/M) de empleado: ")
vSalario=float(input("Entre salario de empleado: "))
```

Insertando registros

```
sql = "INSERT INTO empleados(nombres, \
    apellidos, edad, sexo, salario) \
    VALUES ('%s', '%s', '%d', '%c', '%d' )" % \
    (vNombre, vApellido, vEdad, vSexo, vSalario)
```

try:

```
    cursor.execute(sql)
    # Hace commit a los cambios en la BD
    db.commit()
```

except:

```
    # Hacer rollback si hay algun error
    db.rollback()
```

```
db.close()
```

Lectura de datos

- La lectura implica realizar un query en la BD y luego "alimentar" al programa con los resultados. Para eso se puede usar:
- `fetchone()`: alimenta la siguiente fila del conjunto de resultados del query. Un conjunto de resultados de query es un objeto resultado de un execute hecho a un objeto cursor.
- `fetchall()`: lee todas las filas del conjunto de resultados del query. Si ya se han leído 1 o más filas usando el `fetchone()`, entonces solo lee las filas restantes
- `rowcount`: es un atributo de solo lectura, que devuelve el número de filas ocasionadas por el execute.

Ejemplo

```
import pymysql
db = pymysql.connect("localhost","root","password","testdb" )
cursor = db.cursor()

# Preparar el query SQL de empleados con salario > 1000.
sql = "SELECT * FROM empleados \
      WHERE salario > '%d'" % (1000)
try:
    cursor.execute(sql)
    # Alimentar las filas en una lista de listas.
    results = cursor.fetchall()
```

Ejemplo

```
for fila in results:    # aun en el try
    fname = fila[0]
    lname = fila[1]
    age = fila[2]
    sex = fila[3]
    income = fila[4]
    print ("Nombre=%s, Apellido=%s, Edad=%d, Sexo=%s, Salario=%d" % \
           (fname, lname, age, sex, income ))
except:
    print ("Error: no se pudo leer data")

db.close()
```


Modificando datos

```
import pymysql
db = pymysql.connect("localhost","root","password","testdb" )
cursor = db.cursor()

# aumentar 1 año de edad a los varones
sql = "UPDATE empleados SET edad = edad + 1 WHERE sexo = '%c'" % ('M')

try:
    cursor.execute(sql)
    db.commit()
except:
    db.rollback()

db.close()
```

Borrando registros

```
import pymysql
db = pymysql.connect("localhost","root","password","testdb" )
cursor = db.cursor()

# borrar mayores de 50 años
sql = "DELETE FROM empleados WHERE edad > '%d'" % (50)
try:
    cursor.execute(sql)
    db.commit()
except:
    db.rollback()

db.close()
```

Transacciones

- Son un mecanismo que asegura la consistencia de los datos. Tienen 4 propiedades:
- Atomicidad: o se completa o no pasa nada (todo o nada).
- Consistencia: A transaction must start in a consistent state and leave the system in a consistent state.
- Isolation (Aislamiento): Una transacción no afecta a otra, ni siquiera si se hacen "a la vez". Si no se puede, solo una de las dos transacciones se efectúa en un momento dado.
- Durabilidad: Una vez confirmada una transacción, los efectos son persistentes, aún si se cae el sistema.

Commit y Rollback

- Commit es una operación que da "luz verde" a la base de datos para finalizar los cambios sin que sea posible luego una vuelta atrás.
- Rollback: si uno no está satisfecho con algún cambio y se quiere revertir (deshacer) esos cambios completamente, se usa rollback.
- <execute1><execute2><commit><execute3><commit><execute4><execute5><rollback>

Desarrollo de aplicaciones web

- En lenguajes como Python, PHP y otros hay dos posibles rutas para hacer aplicaciones web:
 - Usando CGI (import cgi) y fabricando los web pages y web forms “a mano”, los queries SQL “a mano”, etc.
 - Usar un framework de desarrollo web (una librería para hacer aplicaciones web)

Django

- Framework web de alto nivel para Python
- Desarrollo rápido
- Automatizar las tareas repetitivas
- Forzar el seguimiento de buenas prácticas

Instalación

- Se necesita una Base de datos (puede ser SQLite que se instala siempre junto a Python, MySQL, PostgreSQL, etc.) instalada. Y Python.
- Luego instalar Django
- `pip install Django==1.10.5`
- Instalo ok? `c:\>python -m django --version`

Pasos generales

- Crear un proyecto en Eclipse (File, New, Other, PyDev, PyDev Django project) , para el ejercicio de clase, llamemoslo mysite
- Para el ejercicio, asociarlo a una base de datos "pollsdb" en MySQL, en 127.0.0.1:3306, root/password. La BD debe haberse creado previamente, pero puede estar vacia.
- En `__init__.py`, añadir:
 - `import pymysql`
 - `pymysql.install_as_MySQLdb()`

Qué hay en el proyecto recién creado?

mysite/

manage.py

mysite/

__init__.py

settings.py

urls.py

wsgi.py

Contenido inicial del proyecto

- Directorio raíz mysite : contenedor del proyecto.
- manage.py: utilidad de línea de comando que permite interactuar con el proyecto de varias maneras.
- Directorio interno mysite: paquete del proyecto. Es el nombre que se usará para importar archivos.
- mysite/__init__.py: "programa principal" del paquete. Generalmente no hace nada, pero lo vamos a usar para cambiar "a mano / a la mala" cierto funcionamiento del Django.
- mysite/settings.py: configuración del proyecto.
- mysite/urls.py: declaraciones de URL del proyecto; una "tabla de contenidos".
- mysite/wsgi.py: instrucciones sobre como hacer que un servidor web que soporte WSGI (Apache por ejemplo) corra la aplicación (modo de producción).

Más pasos

- Probemos que corre:
 - `C:\> python manage.py runserver`
- Abrir `http://127.0.0.1:8000/`
- Dentro del proyecto, crear un "PyDev package", en este ejemplo llamemoslo polls (aplicación para manejar encuestas)

Contenido de polls

__init__.py

admin.py

apps.py

migrations/

 __init__.py

models.py

tests.py

views.py

Contenido de polls

- `models.py`: Modelos de datos
- `views.py`: funciones para visualizar datos (vistas)
- `tests.py`: testing unitario
- Qué falta?
- `urls.py`: configuración de URLs (“URLconf”). Hay que crearlo a mano.

Ejemplo de vista básica

ESCRIBIR EN views.py

```
from django.http import HttpResponse
```

```
def index(request):
```

```
    return HttpResponse("Hola mundo.")
```

Crear /polls/urls.py

```
from django.conf.urls import url
```

```
from . import views
```

```
urlpatterns = [  
    url(r'^$', views.index, name='index'),  
]
```

En mysite/urls.py

```
from django.conf.urls import include, url
from django.contrib import admin
```

```
urlpatterns = [
    url(r'^polls/', include('polls.urls')),
    url(r'^admin/', admin.site.urls),
]
```


OJO

- Entre a `http://localhost:8000/polls/`
- Que tiene cada elemento de `urlpatterns`?
 - Expresión regular de texto
 - “Página web asociada” (en realidad, una función vista que crea páginas web)

Más pasos

- En settings.py (de mysite), definir que aplicaciones preinstaladas y nuevas (INSTALLED_APPS = [...]) y qué BD se usará:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'pollsdb',  
        'USER': 'root',  
        'PASSWORD': 'password',  
        'HOST': '127.0.0.1',  
        'PORT': '3306',  
    }  
}
```

- Crear las tablas de las aplicaciones (Click izquierdo en el proyecto, Django, Migrate)

Acceso a BD con Django

- Django usa object-relational mapping (ORM); el programador define modelos para las tablas de su BD, desde los que objetos Python pueden hacer queries y updates a la BD
- Librerías internas de Django y otros generan el código fuente específico a la BD. Los resultados generados se presentan con conjuntos de resultados (listas de listas)

En polls/models.py

```
from django.db import models
```

```
class Question(models.Model):
```

```
    question_text = models.CharField(max_length=200)
```

```
    pub_date = models.DateTimeField('date published')
```

```
class Choice(models.Model):
```

```
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
```

```
    choice_text = models.CharField(max_length=200)
```

```
    votes = models.IntegerField(default=0)
```

Explicación

- Dos clases de objetos han sido definidos. Cada uno con atributos
- Pero lo que se guardan en MySQL son tablas
- Crear tablas que reflejen esos objetos
- Añadir esta aplicación “a las preinstaladas”, en `mysite/settings.py`:

```
INSTALLED_APPS = [  
    'polls.apps.PollsConfig',    ... ..
```

Más pasos

- `C:\>python manage.py makemigrations polls`
- (crea scripts ara crear las tablas Question, Choice)
- `C:\>python manage.py migrate`
- (ejecuta esos scripts)

Mejorando los modelos de datos

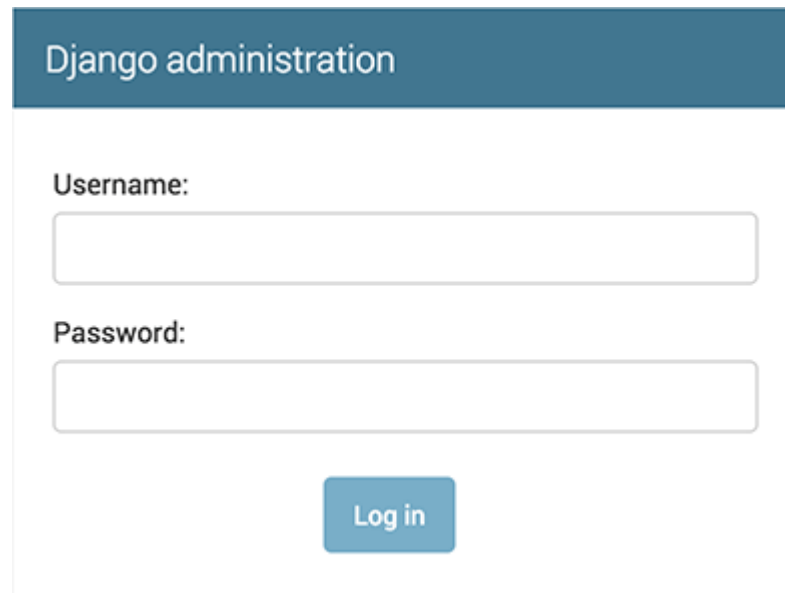
```
import datetime
from django.db import models
from django.utils.encoding import python_2_unicode_compatible
from django.utils import timezone

class Question(models.Model):
    # ...
    def __str__(self):
        return self.question_text
    def was_published_recently(self):
        now = timezone.now()
        return now - datetime.timedelta(days=1) <= self.pub_date <= now

class Choice(models.Model):
    # ...
    def __str__(self):
        return self.choice_text
```

Más pasos

- Crear un usuario administrador de la aplicación (en línea de comandos `python manage.py createsuperuser`)
- Entrar a `http://127.0.0.1:8000/admin/`



The image shows the Django administration login interface. It features a dark blue header with the text "Django administration". Below the header, there are two input fields: "Username:" and "Password:". The "Username:" field is a simple text input, while the "Password:" field is a password input with a small eye icon to toggle visibility. Below these fields is a blue "Log in" button.

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

[+ Add](#) [Change](#)

Users

[+ Add](#) [Change](#)

Recent Actions

My Actions

None available

Más pasos

- Incluir la interfase de Question en la web administrativa
- polls/admin.py:

```
from django.contrib import admin
```

```
from .models import Question
```

```
admin.site.register(Question)
```

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	✎ Change
Users	+ Add	✎ Change
POLLS		
Questions	+ Add	✎ Change

Recent Actions

My Actions

None available

Change question

HISTORY

Question text:

What's up?

Date published:

Date:

2015-09-06

Today | 

Time:

21:16:22

Now | 

Delete

Save and add another

Save and continue editing

SAVE

Más pasos

- Escribir más vistas, polls/views.py:

```
def detail(request, question_id):  
    return HttpResponse("You're looking at question %s." % question_id)  
  
def results(request, question_id):  
    response = "You're looking at the results of question %s."  
    return HttpResponse(response % question_id)  
  
def vote(request, question_id):  
    return HttpResponse("You're voting on question %s." % question_id)
```

Más pasos

- En polls/urls.py

```
from django.conf.urls import url
from . import views
```

```
urlpatterns = [
    # ex: /polls/
    url(r'^$', views.index, name='index'),
    # ex: /polls/5/
    url(r'^(?P<question_id>[0-9]+)/$', views.detail, name='detail'),
    # ex: /polls/5/results/
    url(r'^(?P<question_id>[0-9]+)/results/$', views.results, name='results'),
    # ex: /polls/5/vote/
    url(r'^(?P<question_id>[0-9]+)/vote/$', views.vote, name='vote'),
]
```

Más pasos

```
# EN polls/views.py
from django.http import HttpResponse
from .models import Question
from django.shortcuts import render

def index(request):
    latest_question_list = Question.objects.order_by('-
pub_date')[:5]
    context = {'latest_question_list': latest_question_list}
    return render(request, 'polls/index.html', context)
```

Más pasos

- En polls/templates/polls/index.html

```
{% if latest_question_list %}
```

```
<ul>
```

```
{% for question in latest_question_list %}
```

```
<li><a href="/polls/  
{{ question.id }}/">{{ question.question_text }}</a></li>
```

```
{% endfor %}
```

```
</ul>
```

```
{% else %}
```

```
<p>No polls are available.</p>
```

```
{% endif %}
```


Más pasos

- En polls/templates/polls/detail.html

```
<h1>{{ question.question_text }}</h1>
```

```
<ul>
```

```
{% for choice in question.choice_set.all %}
```

```
    <li>{{ choice.choice_text }}</li>
```

```
{% endfor %}
```

```
</ul>
```

Mejora el anterior

```
<h1>{{ question.question_text }}</h1>
```

```
{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}
```

```
<form action="{% url 'polls:vote' question.id %}" method="post">
```

```
{% csrf_token %}
```

```
{% for choice in question.choice_set.all %}
```

```
    <input type="radio" name="choice" id="choice{{ forloop.counter }}"
value="{{ choice.id }}" />
```

```
    <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br />
```

```
{% endfor %}
```

```
<input type="submit" value="Vote" />
```

```
</form>
```

Más pasos: votar en la encuesta

- En polls/views.py

```
from django.shortcuts import get_object_or_404, render
from django.http import HttpResponseRedirect, HttpResponse
from django.urls import reverse
```

```
from .models import Choice, Question
```

```
# ...
```

```
def vote(request, question_id):
```

```
    question = get_object_or_404(Question, pk=question_id)
```

```
    try:
```

```
        selected_choice = question.choice_set.get(pk=request.POST['choice'])
```

```
    except (KeyError, Choice.DoesNotExist):
```

```
return render(request, 'polls/detail.html', {
    'question': question,
    'error_message': "You didn't select a choice.",
})
```

else:

```
    selected_choice.votes += 1
    selected_choice.save()
    # Always return an HttpResponseRedirect after successfully dealing
    # with POST data. This prevents data from being posted twice if a
    # user hits the Back button.
    return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))
```

def results(request, question_id):

```
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/results.html', {'question': question})
```

Más pasos

- EN polls/templates/polls/results.html

```
<h1>{{ question.question_text }}</h1>
```

```
<ul>
```

```
{% for choice in question.choice_set.all %}
```

```
    <li>{{ choice.choice_text }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>
```

```
{% endfor %}
```

```
</ul>
```

```
<a href="{% url 'polls:detail' question.id %}">Vote again?</a>
```

Más pasos

- En polls/admin.py

```
from django.contrib import admin
```

```
from .models import Choice, Question
```

```
# ...
```

```
admin.site.register(Choice)
```

[Home](#) › [Polls](#) › [Choices](#) › Add choice

Add choice

Question:

Choice text:

Votes: