



## CARGA DE ARCHIVOS



## MODELO

- Dentro de la **app** de registros crea un nuevo modelo Archivos con los siguientes atributos:
  - **id** (llave primaria auto incremental).
  - **título** (tipo char de longitud 100).
  - **archivo** (este atributo permitirá guardar un archivo el cual se enviará a una carpeta “archivos” dentro de nuestra carpeta de media donde se almacenan las imágenes)
    - **archivo = models.FileField(upload\_to="archivos", null=True, blank=True)**
  - **created**
  - **Updated**
  - Agrega al modelo la clase Meta y el método `__str__` para personalizar su vista en el administrador.

## FORMULARIO

El proceso de carga de archivos, lo realizaremos desde una página html, por lo que debemos preparar el archivo de form para la validación de los valores que se enviarán del formulario al modelo:

```
from .models import Archivos
from django.forms import ModelForm, ClearableFileInput

class CustomClearableFileInput(ClearableFileInput):
    template_with_clear = '<br> <label
for="% (clear_checkbox_id)s">% (clear_checkbox_label)s</label>
%(clear)s'
```

**ClearableFileInput:** Widget de los campos FileField, permite que al subir un archivo a un formulario y luego se edita se visualizarán dos campos más. El primero un checkbox de nombre clear, que se encarga de borrar de la base de datos el archivo, y el segundo campo es un input file que permite modificar el archivo que uno subió.

**registros/forms.py**

## FORM

```
class FormArchivos(ModelForm):  
    class Meta:  
        model = Archivos  
        fields = ('titulo', 'descripcion', 'archivo')  
        widgets = {  
            'archivo': CustomClearableFileInput  
        }
```

**registros/forms.py**

VISTA

```
from .models import Archivos  
from .forms import FormArchivos  
from django.contrib import messages
```

**registros/views.py**

## VISTA

## registros/views.py

```
def archivos(request):
    if request.method == 'POST':
        form = FormArchivos(request.POST, request.FILES)
        if form.is_valid():
            titulo = request.POST['titulo']
            descripcion = request.POST['descripcion']
            archivo = request.FILES['archivo']
            insert = Archivos(titulo=titulo, descripcion=descripcion,
                              archivo=archivo)
            insert.save()
            return render(request, "registros/archivos.html")
        else:
            messages.error(request, "Error al procesar el formulario")
    else:
        return render(request, "registros/archivos.html", {'archivo': Archivos})
```



URL

```
path('subir', views_registros.archivos, name="Subir"),
```

**urls.py**

# HTML

- Crea archivos.html
- Copia el código de contactos.html y pega en archivos.html

```
▼ templates / registros
  <> archivos.html
  <> confirmarEliminacion.html
  <> consultaContacto.html
  <> consultas.html
  <> contacto.html
  <> formEditarComentario.html
  <> principal.html
```



# HTML

```
{% extends "inicio/encabezado.html" %}

{% block titulo%} CARGAR ARCHIVO {%endblock%}

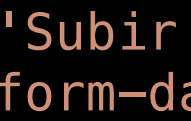
{% block encabezado%}

<!--Encabezdo -->
<div class="page-header">
  <div class="page-title">
    <h3>Envia tu documento
    <small>...</small></h3>
  </div>
</div>
<!-- /Encabezado-->
```

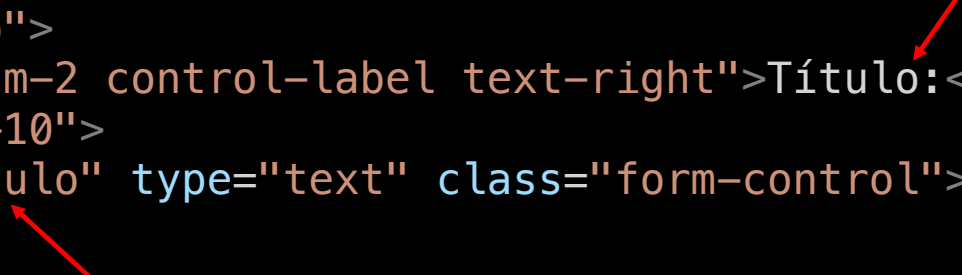
**archivos.html**

## HTML

```
<form class="form-horizontal" action="{% url 'Subir'%}"  
method="POST" role="form" enctype="multipart/form-data">
```



```
<div class="form-group">  
  <label class="col-sm-2 control-label text-right">Título:</label>  
  <div class="col-sm-10">  
    <input name="titulo" type="text" class="form-control">  
  </div>  
</div>
```



**archivos.html**

## HTML

```
<div class="form-group">
  <label class="col-sm-2 control-label text-right">Descripción:</label>
  <div class="col-sm-10">
    <textarea name="descripcion" cols="50" rows="10" class="form-control">
    </textarea>
  </div>
</div>
```

```
<div class="form-group">
  <label class="col-sm-2 control-label text-right">Archivo:</label>
  <div class="col-sm-10">
    <input type="file" name="archivo" />
  </div>
  <button type="button" class="btn btn-primary">Agregar
</div>
```

**archivos.html**

# HTML

- Ejecutamos servidor y accedemos a la ruta:
  - `http://127.0.0.1:8000/subir`

Envia tu documento

...

Principal /subir

Título: Material

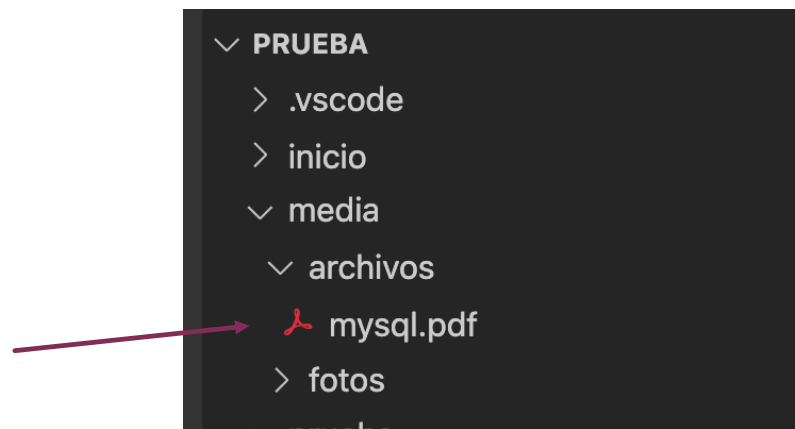
Descripción: Sesión 21 de julio

Archivo:  mysql

Enviar

# HTML

- Si todo es correcto podrás ver el archivo dentro de la carpeta media/archivos.
- Recuerda que media es la carpeta que fue registrada como ubicación de archivos en la practica de carga de imagen.

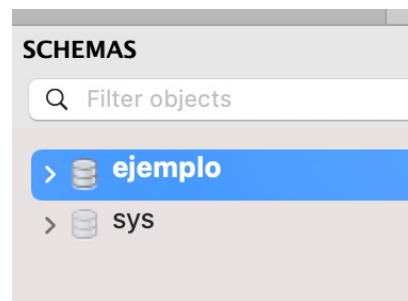




CAMBIANDO EL SGBD

## TRABAJANDO CON MYSQL

- **PASO 1:** Si no cuentas ya con el, descargar e instalar MySQL en tu equipo: <https://dev.mysql.com/downloads/installer/>
- **PASO 2:** Crea una base de datos de nombre **ejemplo**



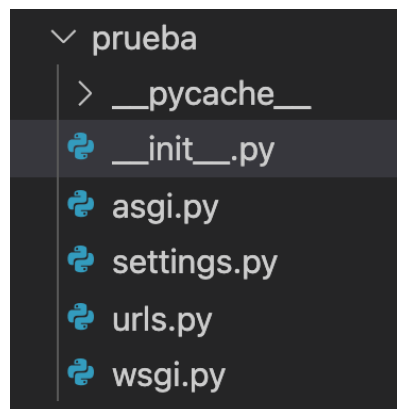
## CONFIGURACIÓN DJANGO CON MYSQL

- **PASO 3:** Instalamos el cliente de MySQL para Python y la librería cryptography encargada de encriptar el acceso a la bd :
  - **pip install pymysql**
  - **pip install cryptography**



## CONFIGURACIÓN DJANGO CON MYSQL

- **PASO 4:** Importamos el cliente en nuestra pymysql en el archivo `__init__` de nuestro proyecto.



```
import pymysql
pymysql.install_as_MySQLdb()
```

**`__init__.py`**

## CONFIGURACIÓN DJANGO CON MYSQL

- **PASO 5:** Configurar la conexión de la aplicación con la base de datos. Ubica la sección de DATABASES en el archivo settings y comenta:

```
'''DATABASES = {  
    'default': {  
        'ENGINE':  
        'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'db.sqlite3',  
    }  
}'''
```

**settings.py**

## CONFIGURACIÓN DJANGO CON MYSQL

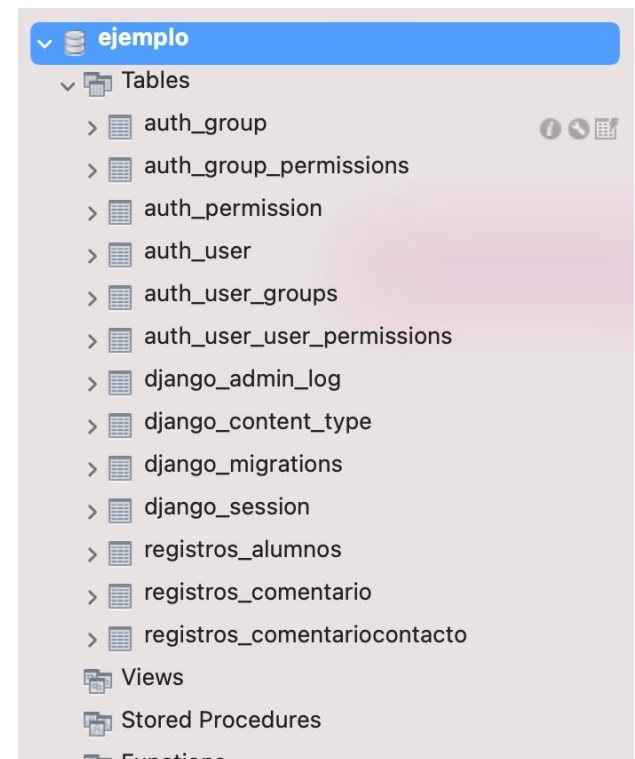
- **PASO 7:** Colocaremos nuestra conexión a mysql.

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'ejemplo',  
        'USER': 'tuUsuario',  
        'PASSWORD': 'tuContraseña',  
        'HOST': 'localhost',  
        'PORT': '3306',  
    }  
}
```

**settings.py**

## CONFIGURACIÓN DJANGO CON MYSQL

- **PASO 8:**Aplicamos las migraciones de los modelos existentes:
  - `python manage.py makemigrations`
  - `python manage.py migrate`
- Ingresa a tu base de datos de ejemplo y verás que se han creado las tablas correspondiente a los modelos del proyecto.



## CONFIGURACIÓN DJANGO CON MYSQL

- **PASO 9:** Prueba tu aplicación, ejecuta tu servidor.
  - **Nota:** Recuerda que es una nueva BD, la base de datos estará vacía, si deseas ingresar al admin debes crear un usuario.
  - Para probar funcionamiento probaremos el módulo de comentarios de contacto el cual cuenta con los procesos CRUD completo:

# CONFIGURACIÓN DJANGO CON MYSQL

## ■ Damos de alta un nuevo comentario.

CONTACTANOS

127.0.0.1:8000/contacto/

Principal  
DSM

Principal

Registro

Contacto

Comentarios

### Contactanos

Ingresa tus dudas

Principal /contacto/

Nombre: Elena

Mensaje: Prueba mysql, este es un comentario

Enviar

# CONFIGURACIÓN DJANGO CON MYSQL

- Si todo es correcto la inserción se realiza, nuestro proyecto debe continuar con el mismo funcionamiento anterior.

## COMENTARIOS

Lista de Comentarios

[Principal /consultarComentario/](#)

Usuario	Mensaje		
Elena	Prueba mysql, este es un comentario.	<a href="#">Editar</a>	<a href="#">Eliminar</a>

# CONFIGURACIÓN DJANGO CON MYSQL

- La diferencia radica en que ahora, la información se encuentra en MySQL. Si accedes a tu base de datos de **ejemplo** creada en **MySQL** y consultas la tabla de **comentariocontacto** podrás ver el comentario registrado desde nuestro proyecto.

```
mysql> select * from registros_comentariocontacto;
```

```
+-----+-----+-----+-----+-----+
| id | usuario | mensaje | created |
+-----+-----+-----+-----+
| 1 | Elena | Prueba mysql, este es un comentario. | 2022-07-14 15:27:29.034964 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```



---

## CONSULTAS DIRECTAS CON SQL

- 
- Como migramos a mysql, nuestra base de datos esta vacia, ingresa al módulo de administración y registra los siguientes alumnos:

Foto	Matricula	Nombre	Carrera	Turno
	UTM7785TI	Monica	TI	Vespertino
	UTM1234	Juan	TI	Matutino
	utm2	Ana	BIO	Vespertino
	UTM1234TIC	Juan	TI	Matutino

## CONSULTAS DIRECTAS CON SQL

- Django permite realizar dos formas de consultas SQL:
- **ORM** (Object Relational Mapper): Permite interactuar con nuestra base de datos sin la necesidad de conocer SQL.
- Usando **Manage.raw()**: **raw()** es un método del administrador que se puede utilizar para realizar consultas SQL sin procesar que devuelven instancias de modelo, permite ejecutar SQL personalizado y directo.

## CONSULTAS DIRECTAS CON SQL

```
def consultasSQL(request):  
    alumnos=Alumnos.objects.raw('SELECT  
matricula,nombre, carrera, turno, imagen FROM  
registros_alumnos WHERE carrera="TI" ORDER BY  
turno DESC')  
  
    return  
    render(request,"registros/consultas.html",  
{'alumnos':alumnos})
```

**views.py**

## CONSULTAS DIRECTAS CON SQL

- Agregamos una nueva url:

```
path('consultasSQL', views_registros.consultasSQL, name="sql"),
```

**urls.py**

- Ejecutamos servidor y accedemos a la ruta:
  - <http://127.0.0.1:8000/consultasSQL>

# CONSULTAS DIRECTAS CON SQL

## FieldDoesNotExist at /consultasSQL

Raw query must include the primary key

**Request Method:** GET

**Request URL:** http://127.0.0.1:8000/consultasSQL

**Django Version:** 3.2.4

**Exception Type:** FieldDoesNotExist

**Exception Value:** Raw query must include the primary key

**Exception Location:** /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/django/db/models/query.py, line 1499, in iterator

**Python Executable:** /Library/Frameworks/Python.framework/Versions/3.9/bin/python3

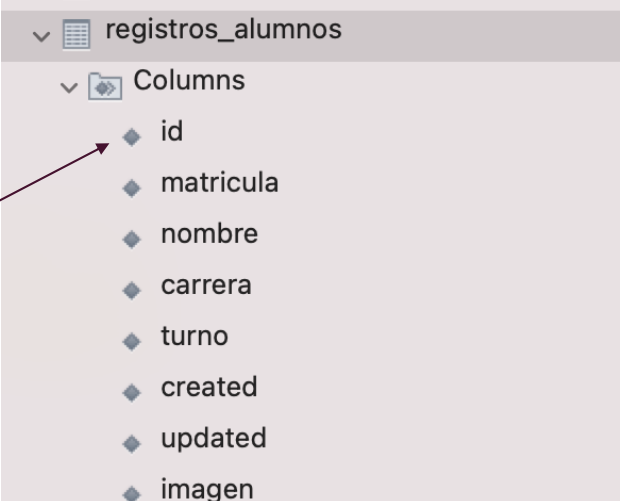
**Python Version:** 3.9.5

**Python Path:** ['/Users/elena/Documents/proyectos Django/prueba',  
'/Library/Frameworks/Python.framework/Versions/3.9/lib/python39.zip',  
'/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9',  
'/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/lib-dynload',  
'/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages']

**Server time:** Thu, 22 Jul 2021 01:25:10 +0000

## CONSULTAS DIRECTAS CON SQL

En este mecanismo de consulta, tiene la restricción de consultar el campo id ya que como identificador se toma como requerido para procesos que puedan ser requeridos en la aplicación. En nuestro caso el modelo de alumno no cuenta con llave primaria, sin embargo si accedemos a la estructura de la tabla observarán que fue agregado automáticamente.



A screenshot of a database interface showing the structure of a table named 'registros\_alumnos'. The table has a 'Columns' section with the following fields: 'id', 'matricula', 'nombre', 'carrera', 'turno', 'created', 'updated', and 'imagen'. A red arrow points to the 'id' field, highlighting it as the primary key.

registros_alumnos
Columns
id
matricula
nombre
carrera
turno
created
updated
imagen

## CONSULTAS DIRECTAS CON SQL

```
def consultasSQL(request):  
    alumnos=Alumnos.objects.raw('SELECT id,  
matricula,nombre, carrera, turno, imagen FROM  
registros_alumnos WHERE carrera="TI" ORDER BY  
turno DESC')  
  
    return  
    render(request,"registros/consultas.html",  
{'alumnos':alumnos})
```

Agrega

**views.py**



## CONSULTAS DIRECTAS CON SQL

- Ejecutamos servidor y accedemos a la ruta:
  - <http://127.0.0.1:8000/consultasSQL>

Foto	Matricula	Nombre	Carrera	Turno
	UTM7785TI	Monica	TI	Vespertino
	UTM1234TIC	Juan	TI	Matutino
	UTM1234	Juan	TI	Matutino

```
SELECT id,matricula,nombre,carrera,turno,imagen FROM registros_alumnos WHERE  
carrera="TI" ORDER BY turno DESC
```



#### Explore the ORM before using raw SQL!

The Django ORM provides many tools to express queries without writing raw SQL. For example:

- The [QuerySet API](#) is extensive.
- You can [annotate](#) and [aggregate](#) using many built-in [database functions](#). Beyond those, you can create [custom query expressions](#).

Before using raw SQL, explore [the ORM](#). Ask on one of [the support channels](#) to see if the ORM supports your use case.



#### Warning

You should be very careful whenever you write raw SQL. Every time you use it, you should properly escape any parameters that the user can control by using [params](#) in order to protect against SQL injection attacks. Please read more about [SQL injection protection](#).

# ORM<sub>vs</sub> SQL

¿Por qué DJANGO apuesta por ORM?

Para terminar tu actividades, lee sobre ORM y SQL, sus ventajas y desventajas.