Dark Academy: aplicación de Python DB API 2.0 DB con MySQL

PARTE 1

Como entrenamiento se adjuntan ejemplos de https://dev.mysql.com/doc/connector-python/en/ (carpeta recursos_mysql) y lo primero que necesitamos en instalar el conector de Python para MySQL con:

```
pip install mysql-connector-python
```

NOTA: si no va es que necesitas instalar el pip (gestor de paquetes Python) https://pip.pypa.io/en/stable/installation/

En el instituto se dispone de un servidor con MySQL8 y solo debes pedir una contraseña de acceso al profesor.

Si vas a trabajar en casa puedes instalarte MySQL8 en tu ordenador, en docker o en una máquina virtual.

Para previo estudio se añaden los siguientes ficheros (carpeta recursos_mysql):

- conectar.py y conectar_logger.py: ejemplos de como conectar a MySQL8 de diversas formas.
- crear_tablas.py: ejemplo de como crear las tablas de la BDA employees.
  Si tras crear las tablas queremos poblar las tablas:
  Descomprimimos con el botón derecho y nos crea la carpeta employees_db.
  Abrimos dicha carpeta con un terminal y cargamos los datos de cada una de las tablas con el terminal:

```
$> mysql -u root -p employees < load_employees.dump
$> mysql -u root -p employees < load_titles.dump
$> mysql -u root -p employees < load_departments.dump
$> mysql -u root -p employees < load_salaries.dump
$> mysql -u root -p employees < load_dept_emp.dump
$> mysql -u root -p employees < load_dept_manager.dump
```

  ```
  NOTA: para que funcione el ejemplo anterior hay que añadir al principio de cada fichero.dump: use employees
  ```

- insertar_datos.py: distintas formas de actualizar datos.
- consultar_datos.py: ejemplo de consultas de datos.

Prueba los distintos códigos y verás que es muy parecido a lo que se vio con SQLite3 solo que en este caso utilizamos el dialecto de SQL de MySQL que ya se estudió en BDA de 1DAM.

Se pide (hasta 5 puntos)

1. Conectar al servidor mysql8 mediante Python

2. Crear la BDA Dark_Academy mediante Python. Si la BDA ya existe (un compañer@ se te ha adelantado) debes crear otra con tu nombre como sufijo ("Dark_Academy" + "_" + nombre). El SQL:

```
create database if not exists Dark_Academy
character set utf8mb4
collate utf8mb4_spanish_ci;
```

3. Con Python selecciona la BDA para trabajar con ella. El SQL:

```
use Dark_Academy;
```

4. Con Python crea las tablas de la BDA. EL SQL:

```
drop table if exists alumnos;
create table alumnos(
	expediente char(8) PRIMARY KEY,
	nombre varchar(30) NOT NULL,
	apellidos varchar(50) NOT NULL
);
drop table if exists modulos;
create table modulos(
	codigo varchar(5) PRIMARY KEY,
	nombre varchar(30) NOT NULL
);
drop table if exists notas;
create table notas(
	expediente char(8),
	codigo varchar(5),
	nota integer unsigned,
	PRIMARY KEY (expediente, codigo),
	constraint fk_expediente
		foreign key (expediente)
		references alumnos(expediente)
		on delete cascade on update cascade,
	constraint fk_codigo
		foreign key (codigo)
		references modulos(codigo)
		on delete cascade on update cascade
);
```

5. Desde Python creamos tres triggers que hagan una auditoría de la tabla notas. Se auditan inserciones, borrados y modificaciones. El SQL:

```
drop table if exists auditoria_notas;
create table auditoria_notas(
   id serial PRIMARY KEY, -- clave autoincremental
   expediente_old char(8), -- expediente viejo
   codigo_old varchar(5), -- módulo viejo
   nota_old integer unsigned, -- nota antigua
   expediente_new char(8), -- expediente nuevo
   codigo_new varchar(5), -- módulo nuevo
   nota_new integer unsigned, -- nota nueva
   usuario varchar(50) not null, -- usuario que hace la modificación
```

```
    cuando datetime not null, -- fecha y hora de la modificación
    operacion enum('insert', 'update', 'delete') not null -- operación DML utilizada
);

delimiter ;
drop trigger if exists auditoria_notas_insert;
create trigger auditoria_notas_insert after insert on notas
for each row
    insert into auditoria_notas
    values(null,null,null,null,new.expediente, new.codigo, new.nota, user(), now(),'insert');
delimiter ;

drop trigger if exists auditoria_notas_update;
create trigger auditoria_notas_update after update on notas
for each row
    insert into auditoria_notas
    values(null, old.expediente, old.codigo, old.nota, new.expediente, new.codigo,
    new.nota, user(), now(), 'update');
delimiter ;

drop trigger if exists auditoria_notas_delete;
create trigger auditoria_notas_delete after delete on notas
for each row
    insert into auditoria_notas
    values(null, old.expediente, old.codigo, old.nota, null,null,null, user(), now(), 'delete');
```

6. Con Python inserta los datos en las tablas. Sería interesante insertar usando una lista de tuplas. El SQL:

```
insert into alumnos values
        ('11111111', 'Alexia', 'Núñez Pérez'),
        ('22222222', 'Rosa', 'Fernández Oliva'),
        ('33333333', 'Peter', 'Linuesa Jiménez'),
        ('44444444', 'Juan Carlos', 'Wesnoth The Second'),
        ('55555555', 'Federico', 'Muñoz Ferrer');

insert into modulos values
        ('QP', 'Quirománcia Práctica'),
        ('MR', 'Mortum Redivivus'),
        ('RF', 'Refactorización Zómbica'),
        ('ARF', 'Ampliación de RF'),
        ('OP', 'Orquestación de Plagas');

insert into notas values
        ('11111111', 'QP', 5),
        ('11111111', 'MR', 7),
        ('11111111', 'RF', 6),
        ('11111111', 'ARF', 9),
        ('11111111', 'OP', 7),
        ('22222222', 'QP', NULL),
        ('22222222', 'MR', 5),
        ('22222222', 'RF', 5),
        ('22222222', 'ARF', 6),
        ('22222222', 'OP', NULL),
```

```
('33333333', 'QP', 9),
('33333333', 'MR', 5),
('33333333', 'RF', 6),
('33333333', 'ARF', 4),
('33333333', 'OP', 6),
('44444444', 'QP', 4),
('44444444', 'MR', 6),
('44444444', 'RF', 8),
('44444444', 'ARF', 6),
('44444444', 'OP', 5),
('55555555', 'QP', 8),
('55555555', 'MR', 4),
('55555555', 'RF', NULL),
('55555555', 'ARF', NULL),
('55555555', 'OP', 4);
```

7. Con Pyhton muestra los datos de todas las tablas incluida la de auditoría. Sería interesante parametrizar el nombre de la tabla en una función. El SQL:

   ```
   select * from alumnos;
   select * from modulos
   select * from notas;
   select * from auditoria_notas;
   ```

8. Con Python diseña una función en el lado del servidor que compruebe si un expediente es correcto. Un expediente correcto debe tener 8 dígitos numéricos. A la función se le pasará un char(8) como parámetro y retornará un boolean (0 o False, 1 o True). El SQL:

   ```
   delimiter //
   create function expediente_correcto(exp char(8)) returns BOOLEAN
   DETERMINISTIC
   NO SQL
   begin
      return (exp regexp '^[0-9]{8}$');
   end//
   delimiter ;
   ```

9. Con Python diseña una función que compruebe si un expediente es correcto utilizando la función del servidor del punto 8.

10. Diseña una función en Python que imprima los datos de los alumnos que pasan a segundo del ciclo 'Artes Oscuras' y también debe imprimir el porcentaje de alumnos que pasan a segundo. Los requisitos que debe cumplir un alumno para pasar a segundo son: tener todos los módulos con nota y que la media de las notas sea mayor o igual a 6. Si un alumno no se ha examinado de un módulo entonces su nota es NULL y, obviamente, ya no pasa a segundo.

Se os proporciona DA_plantilla_alumnos.py con bastantes cosas hechas.

Rúbrica:

Toda la funcionalidad está hecha → 4 puntos

Se envuelven en try … except las cosas críticas (cosas que pueden dar error en mysql) → 1 punto.

PARTE 2

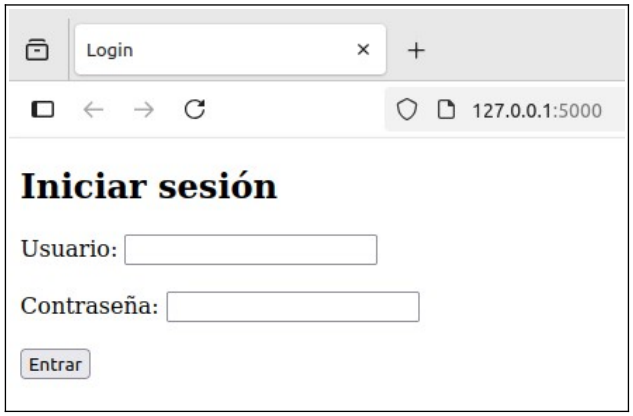
Desarrollo de una App WEB para hacer el CRUD (Create, Read, Update y Delete) sobre las tablas alumnos, modulos y notas. Además se usa una seguridad mínima para acceder a la aplicación (usuario + contraseña)

Tecnologías utilizadas: Python, Flask, Jinja2, HTML y MySQL.

En la carpeta recursos_flask hay un minitutorial de flask y jinja2.

Ejemplo de funcionamiento del código completo:

1. Tras arrancar el servidor de desarrollo hacemos CNTRL+CLICK al enlace que nos ofrece o escribimos 127.0.0.1:5000 en el navegador. Aparece login HTML:

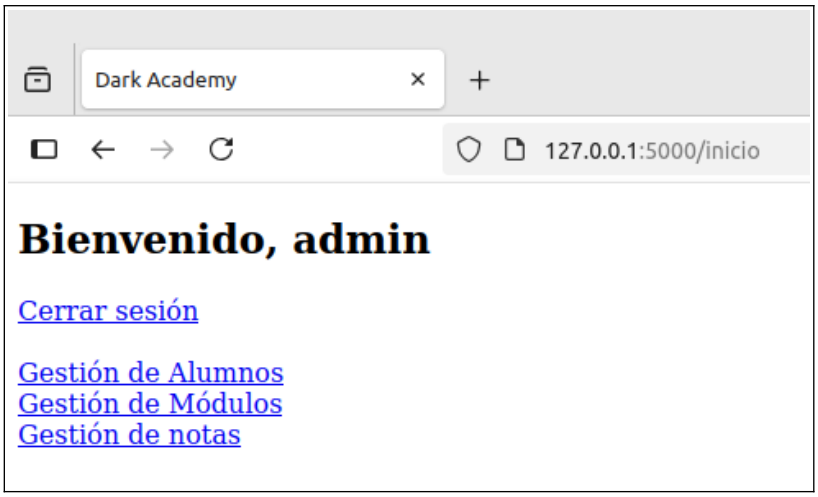


Si el login es correcto nos lleva a inicio.html. Si el login es incorrecto nos lleva a login.html de nuevo (informando de que la contraseña es incorrecta).

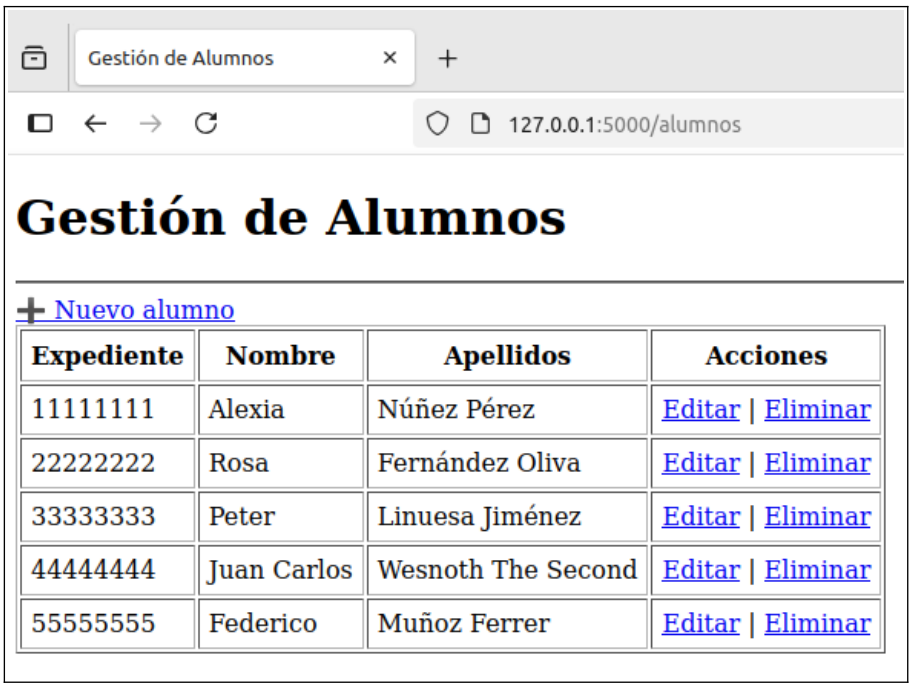
2. En la página inicio.html tenemos las opciones:

Cerrar sesión: nos lleva a login.html

Gestión de Alumnos, de Módulos y de Notas: CRUD de alumnos, módulos y notas respectivamente.



3. Si clickamos a Gestión de Alumnos nos lleva a alumnos.html (en realidad es una composición de alumnos_base + alumnos.html) y desde esta página podemos hacer el CRUD sobre la tabla alumnos.



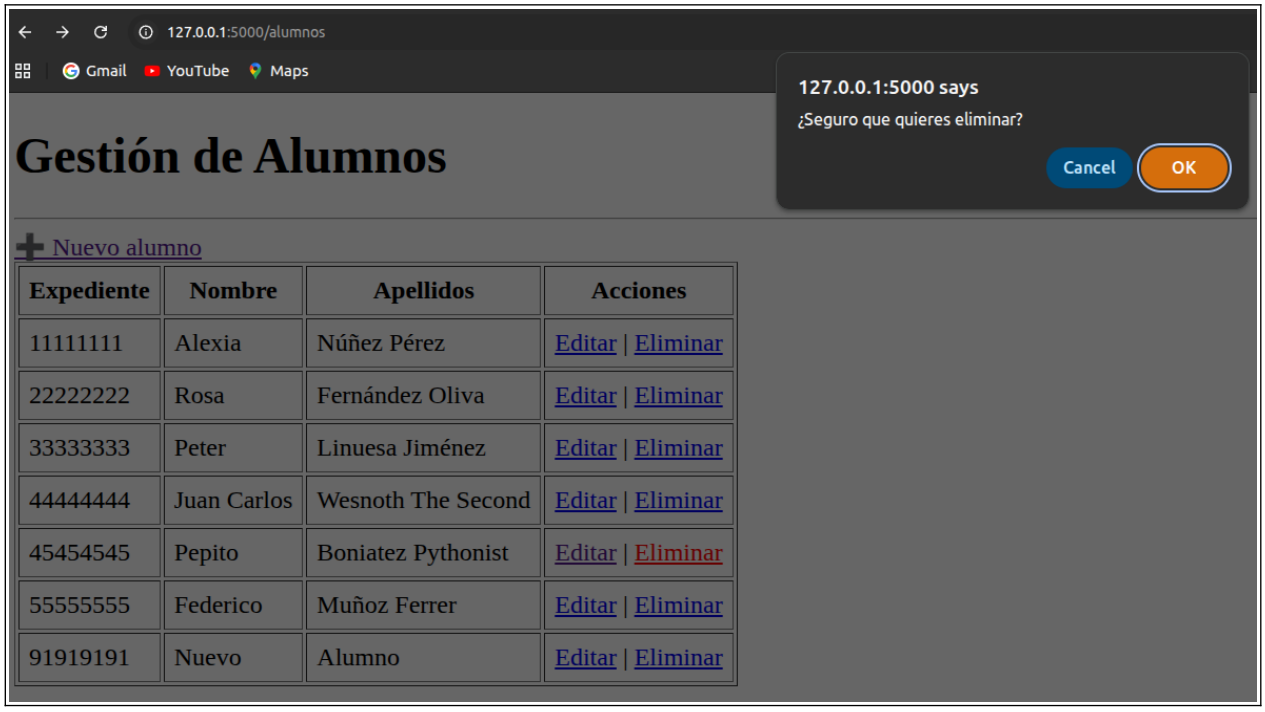
4. Si clickamos Nuevo alumno nos lleva a alumnos_nuevo.html que permite añadir un alumno:

Al clickar Guardar nos devuelve a Gestión de alumnos listando todos los alumnos incluido el que acabamos de añadir.

5. Si desde Gestión de Alumnos clickamos Editar nos lleva a alumnos_editar.html donde podemos cambiar los datos del alumno.

Al clickar Actualizar nos devuelve al listado de Gestión de alumnos donde aparecen los datos del alumno actualizados.

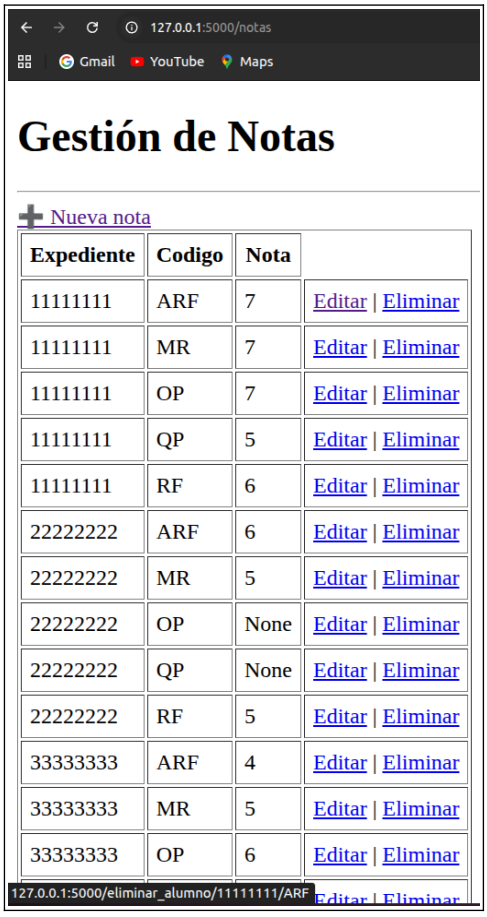
6. Si desde gestión de alumnos clickamos Eliminar



Si le damos a OK nos devuelve a la misma página pero sin el alumno en cuestión.

7. Gestión de Módulos tiene un comportamiento similar:

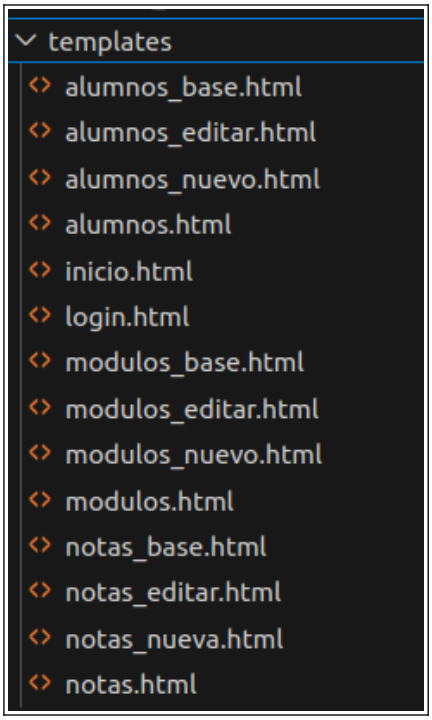
8. Gestión de notas tiene un comportamiento similar:



PERO si nos fijamos en la imagen para editar/borrar notas necesitamos pasar 2 parámetros ya que la clave es compuesta.

Punto de partida

- Se os pasan todas las plantillas HTML con jinja2:



NOTA: como la gestión de modulos y notas en Python la tenéis que implementar vosotr@s. debéis comentar algunas líneas en la plantilla inicio para no tener errores. (como véis se comenta el HTML con <!-- --> y el Jinja2 con {# #})

inicio.html

Dark_academy > templates > inicio.html > ...

```
<!DOCTYPE html>
<html>
<head>
    <title>Dark Academy</title>
</head>
<body>
    <h2>Bienvenido, {{ usuario }}</h2>
    <a href="{{ url_for('logout') }}">Cerrar sesión</a>
    <br><br>
    <a href="{{ url_for('alumnos') }}">Gestión de Alumnos</a><br>
    <!-- <a href="{#{{ url_for('modulos' )}}#}">Gestión de Módulos</a><br> -->
    <!-- <a href="{#{{ url_for('notas') }}#}">Gestión de Notas</a> -->

</body>
</html>
```

- Se os pasa la app web solo funcional para el CRUD de alumnos: app_web_plantilla.py

¿Qué hay que hacer?

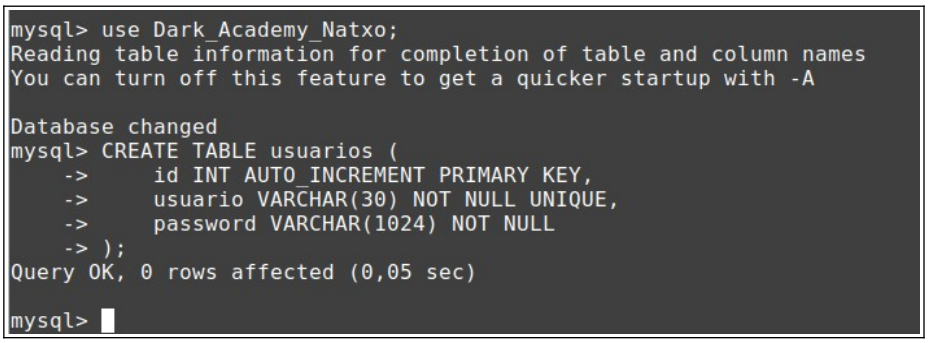
- Implementar el CRUD de modulos y notas (4 puntos)
- Mejorar la seguridad de la app (1 punto)

ANEXO I: seguridad mejorada

La idea es guardar las contraseñas cifradas (hash) en una tabla de la BDA. Lo primero es crear una tabla en la BDA:

```
CREATE TABLE usuarios (
        id INT AUTO_INCREMENT PRIMARY KEY,
        usuario VARCHAR(30) NOT NULL UNIQUE,
        password VARCHAR(1024) NOT NULL
);
```

Esto se puede hacer con Python o directamente mediante mysql-client (más rápido). Ejemplo con mysql-client:



Necesitaremos hacer el siguiente import:

```
from werkzeug.security import generate_password_hash, check_password_hash
```

CONSEJO: os podéis generar una página de registro asociada a la ruta /registro que permita añadir nombres y passwords (hasheadas) en la tabla de forma cómoda.

La página de registro tendrá el siguiente aspecto:

El método asociado a la ruta /registro deberá obtener el usuario y contraseña escritos y aplicar el siguiente algoritmo:

```
...
# Hash seguro
password_hash = generate_password_hash(password)

conn = connect_to_mysql(config)
cursor = conn.cursor()
use_database(cursor, DB_NAME)
```

```
# Verificar que no exista usuario
cursor.execute("SELECT * FROM usuarios WHERE usuario = %s", (usuario,))
existe = cursor.fetchone()

if existe:
        flash("El usuario ya existe. Elige otro nombre.")
        cursor.close()
        conn.close()
        return redirect(url_for("registro"))

# Insertar en BD
cursor.execute(
        "INSERT INTO usuarios (usuario, password) VALUES (%s, %s)",
(usuario, password_hash)
)
...
```

En la ruta de / (login) deberemos comprobar que el usuario está en la tabla usuarios y que el hash de la contraseña que ha introducido coincide con el hash guardado para ese usuario.

```
…

if check_password_hash(user["password"], password):

        session["usuario"] = user["usuario"]
        return redirect(url_for("inicio"))
...
```