INFORME PROYECTO FINAL

BASES DE DATOS I

Gastón Bauer

Juan Ignacio Cama

Sebastián Laguna

Juan Ignacio Rodríguez

Índice

1. Introducción

El presente informe documenta el desarrollo de una API para un sistema de gestión de una base de datos orientada a clases y alumnos en un entorno educativo.

Este proyecto se diseñó para optimizar la asignación de estudiantes a clases, la gestión de instructores y la administración de turnos.

A través de una arquitectura basada en Python y MySQL, se lograron implementar funcionalidades clave como la creación, edición y eliminación de registros. Además, se integraron validaciones para asegurar la consistencia de los datos, respetando las restricciones de claves foráneas.

---

2. Requerimientos

Requerimientos funcionales:

- Crear, editar y eliminar registros de clases, instructores y alumnos.
- Asociar alumnos a clases y turnos, validando que los registros referenciados existan.
- Consultar turnos y horarios disponibles.

Requerimientos no funcionales:

- La aplicación debe garantizar integridad referencial en la base de datos.
- El sistema debe ser accesible a través de una API REST, desarrollada en Python con Flask.
- Optimización en el uso de consultas SQL para minimizar la carga en el servidor.

---

3. Justificación de decisiones de diseño

Elección de MySQL sobre SQL Server: MySQL fue seleccionado por su facilidad de configuración y menor consumo de recursos en comparación con SQL Server, lo que lo hace más accesible para proyectos de desarrollo y entornos pequeños. En términos de compatibilidad, MySQL se integró perfectamente con las herramientas utilizadas (como Docker y Flask), simplificando la orquestación del entorno.

Uso de Flask: fue elegido por ser un microframework ligero y flexible, lo que permitió concentrarse en las necesidades específicas del proyecto sin imponer una estructura rígida. Su facilidad para extender funcionalidades y la compatibilidad con bibliotecas populares (como Blueprint, request, jsonify) lo hicieron ideal para construir una API REST que conectara la lógica de negocio con la base de datos. La curva de aprendizaje de Flask es baja, lo que permitió desarrollar rápidamente funcionalidades clave.

Estructura modular del código: Se optó por una organización modular en capas (controladores, servicios y configuraciones) para separar las responsabilidades dentro del código:

- Los controladores manejan las rutas y solicitudes HTTP.
- Los servicios contienen la lógica de negocio y las interacciones con la base de datos.
- Los archivos de configuración gestionan parámetros reutilizables como las conexiones.

Este enfoque facilita el mantenimiento del proyecto y la reutilización de componentes, además de permitir una evolución escalable en futuros desarrollos.

Validaciones en la base de datos: Se implementaron validaciones directamente en la base de datos mediante restricciones de clave foránea (FOREIGN KEY) y condiciones como NOT NULL. Este enfoque garantiza que los datos se mantengan consistentes, independientemente del cliente que acceda a la base de datos. Complementando las validaciones del backend en Flask, las restricciones a nivel de base de datos actúan como una capa adicional de seguridad y robustez.

Consideraciones de seguridad: utilizamos ‘%s’ como medida de protección frente a SQL Injection entre otras cosas. %s funciona como marcador de posición para luego pasar los valores como parámetros y el motor de la base de datos maneja el escape de los valores automáticamente. Esto protege tu aplicación de ataques de SQL

Injection, ya que los datos del usuario no se insertan directamente en la consulta como texto plano. Es una práctica recomendada antes de realizar la query como una string e irle concatenando los valores. Además nos permite que la lógica SQL y los datos sean tratados por separado, mejorando la claridad del código y simplificando la manipulación de los datos sin interferir con la estructura de la consulta.

---

4. Fundamento teórico

Bases de datos relacionales: Las bases de datos relacionales son sistemas que organizan la información en tablas relacionadas entre sí mediante claves primarias y foráneas. Consideramos que para este proyecto se opta por utilizar este tipo de base para implementar una estructura que garantiza la consistencia y la integridad de los datos mediante restricciones de clave foránea.

Flask como microframework: Flask es un microframework de Python diseñado para el desarrollo de aplicaciones web ligeras. Su flexibilidad permitió implementar una API RESTful que conecta la lógica de negocio con la base de datos, facilitando la interacción entre el cliente y el servidor.

Consultas SQL optimizadas: Las consultas SQL optimizadas son esenciales para garantizar el rendimiento del sistema. En este proyecto, se utilizaron operaciones JOIN para relacionar múltiples tablas, y funciones como SUM y COUNT para calcular métricas clave, como ingresos y popularidad.

Modelado de datos: El modelo de datos fue diseñado para representar las entidades principales del sistema: clases, alumnos, instructores, y turnos. La relación entre estas tablas se implementó mediante claves foráneas, permitiendo consultas complejas como la obtención de la clase más rentable o el turno más popular.

Docker: Docker es una herramienta que permite empaquetar aplicaciones y sus dependencias en contenedores. Los contenedores garantizan que el software se ejecute de manera uniforme en cualquier entorno, ya sea en desarrollo, pruebas o producción. Esto reduce problemas relacionados con diferencias en configuraciones del sistema operativo o en dependencias.

Docker Compose es una herramienta que permite definir y administrar múltiples contenedores como servicios interdependientes en un archivo docker-compose.yml. La implementación volúmenes nos permite persistir datos y compartir archivos entre el host y los contenedores. Los mismos mapean los archivos init.sql e init_data.sql al directorio /docker-entrypoint-initdb.d/ del contenedor para que MySQL ejecuta automáticamente estos scripts durante la inicialización, creando tablas, relaciones y datos predefinidos.

---

5. Bitácora

Fecha	**Actividad**	**Detalles**
25/10/2024	Estudio de las distintas alternativas para realizar el proyecto.	Se analiza las alternativas de bases de datos relacionales y la implementación de Docker.
01/11/2024	Inicialización del proyecto	Creación del proyecto en Python y la configuración de la base de datos y Docker.
05/11/2024	Decisiones de diseño	Elegimos Flask como framework para facilitar la creación de la API.
08/11/2024	Implementación configuraciones iniciales	Se aprovecha la utilización de Docker para que al momento de levantar nuestro contenedor tengamos datos iniciales .
15/11/2024	Ruteo de la API	Se empieza a mapear la API, los distintos endpoints requeridos para cumplir con los objetivos.
15/11/2024	Primeras query	Comenzamos a diseñar las consultas básicas para los endpoints
19/11/2024	Query más complejas	Implementación de consultas más complejas
23/11/2024	API completa	Se implementan los últimos métodos necesarios para el análisis de las métricas del negocio. A partir de este punto se empieza a probar el funcionamiento total del proyecto

6. Conclusiones

El proyecto permitió la implementación de un sistema funcional para la gestión de datos relacionados con clases, alumnos e instructores. Durante el desarrollo, se aprendió la importancia de validar entradas tanto en el backend como en la base de datos, así como la utilidad de un diseño modular. Como mejoras futuras, se recomienda seguir mejorando el manejo de los datos que recibimos como parámetros en los distintos endpoints, en conjunto con la implementación de un sistema de manejo de errores que permita al usuario tener un mayor conocimiento en caso de hacer requests inválidas para nuestro sistema.

---

7. Bibliografía

MySQL Documentation: https://dev.mysql.com/doc/

Flask Documentation: https://flask.palletsprojects.com/

Python Official Documentation: https://docs.python.org/

Stack Overflow: Consultas específicas sobre errores y dudas generales.

---

8. Anexo

Creación de tablas:

```
CREATE TABLE login (
  correo VARCHAR(255) PRIMARY KEY,
  contrasena VARCHAR(255) NOT NULL
);

CREATE TABLE actividades (
  id INT AUTO_INCREMENT PRIMARY KEY,
  descripcion VARCHAR(255) NOT NULL,
  costo DECIMAL(10, 2) NOT NULL
);

CREATE TABLE equipamiento (
  id INT AUTO_INCREMENT PRIMARY KEY,
  id_actividad INT,
  descripcion VARCHAR(255) NOT NULL,
  costo DECIMAL(10, 2) NOT NULL,
  FOREIGN KEY (id_actividad) REFERENCES actividades(id)
);

CREATE TABLE instructores (
  ci INT PRIMARY KEY,
```

```
  nombre VARCHAR(255) NOT NULL,
  apellido VARCHAR(255) NOT NULL
);

CREATE TABLE turnos (
  id INT AUTO_INCREMENT PRIMARY KEY,
  hora_inicio TIME NOT NULL,
  hora_fin TIME NOT NULL
);

CREATE TABLE alumnos (
  ci INT PRIMARY KEY,
  nombre VARCHAR(255) NOT NULL,
  apellido VARCHAR(255) NOT NULL,
  fecha_nacimiento DATE NOT NULL,
  telefono VARCHAR(15),
  mail VARCHAR(255)
);

CREATE TABLE clase (
  id INT AUTO_INCREMENT PRIMARY KEY,
  ci_instructor INT,
  id_actividad INT,
  id_turno INT,
  dictada BOOLEAN NOT NULL,
  FOREIGN KEY (ci_instructor) REFERENCES instructores(ci),
  FOREIGN KEY (id_actividad) REFERENCES actividades(id),
  FOREIGN KEY (id_turno) REFERENCES turnos(id)
);

CREATE TABLE alumno_clase (
  id_clase INT,
  ci_alumno INT,
  id_equipo INT,
  PRIMARY KEY (id_clase, ci_alumno, id_equipo),
  FOREIGN KEY (id_clase) REFERENCES clase(id),
  FOREIGN KEY (ci_alumno) REFERENCES alumnos(ci),
```

```
   FOREIGN KEY (id_equipo) REFERENCES equipamiento(id)
);
```

Datos iniciales básicos:

```
-- Insertar actividades
INSERT INTO actividades (descripcion, costo) VALUES ('snowboard', 100.00);
INSERT INTO actividades (descripcion, costo) VALUES ('ski', 120.00);
INSERT INTO actividades (descripcion, costo) VALUES ('moto de nieve', 150.00);

-- Insertar turnos
INSERT INTO turnos (hora_inicio, hora_fin) VALUES ('09:00:00', '11:00:00');
INSERT INTO turnos (hora_inicio, hora_fin) VALUES ('12:00:00', '14:00:00');
INSERT INTO turnos (hora_inicio, hora_fin) VALUES ('16:00:00', '18:00:00');

-- Insertar equipamiento
INSERT INTO equipamiento (id_actividad, descripcion, costo) VALUES (1, 'Tabla de
snowboard', 50.00);
INSERT INTO equipamiento (id_actividad, descripcion, costo) VALUES (1, 'Casco de
snowboard', 15.00);
INSERT INTO equipamiento (id_actividad, descripcion, costo) VALUES (1, 'Botas de
snowboard', 30.00);
INSERT INTO equipamiento (id_actividad, descripcion, costo) VALUES (1, 'Guantes
termicos', 10.00);
INSERT INTO equipamiento (id_actividad, descripcion, costo) VALUES (2, 'Esquis',
60.00);
INSERT INTO equipamiento (id_actividad, descripcion, costo) VALUES (2, 'Casco de
ski', 15.00);
INSERT INTO equipamiento (id_actividad, descripcion, costo) VALUES (2, 'Botas de
ski', 35.00);
INSERT INTO equipamiento (id_actividad, descripcion, costo) VALUES (2, 'Gafas de
ski', 20.00);
INSERT INTO equipamiento (id_actividad, descripcion, costo) VALUES (3, 'Casco para
moto de nieve', 20.00);
INSERT INTO equipamiento (id_actividad, descripcion, costo) VALUES (3, 'Guantes
para moto de nieve', 12.00);
```

ABM Instructores:

INSERT INTO instructores (ci, nombre, apellido) VALUES (%s, %s, %s)
UPDATE instructores SET nombre = %s, apellido = %s WHERE ci = %s
DELETE FROM instructores WHERE ci = %s

ABM Turnos:

SELECT * FROM turnos
INSERT INTO turnos (hora_inicio, hora_fin) VALUES (%s, %s)
UPDATE turnos SET hora_inicio = %s, hora_fin = %s WHERE id = %s
DELETE FROM turnos WHERE id = %s

ABM Alumnos:

INSERT INTO alumnos (ci, nombre, apellido, fecha_nacimiento, mail, telefono)
VALUES (%s, %s, %s, %s, %s, %s)
UPDATE alumnos SET nombre = %s, apellido = %s, fecha_nacimiento = %s, mail = %s,
telefono = %s WHERE ci = %s
DELETE FROM alumnos WHERE ci = %s

Métricas:

SELECT id_clase, *COUNT*(ci_alumno) AS cantidad_alumnos FROM alumno_clase
GROUP BY id_clase ORDER BY cantidad_alumnos DESC LIMIT 1;
SELECT
  ac.id_clase,
  SUM(a.costo + e.costo) AS ingresos_totales
FROM
  alumno_clase ac
JOIN
  clase c ON ac.id_clase = c.id
JOIN
  actividades a ON c.id_actividad = a.id
JOIN
  equipamiento e ON ac.id_equipo = e.id
GROUP BY

```
  ac.id_clase
ORDER BY
  ingresos_totales DESC
LIMIT 1;
SELECT
  c.id_turno,
  COUNT(c.id) AS cantidad_clases
FROM
  clase c
WHERE
  c.dictada = 1
GROUP BY
  c.id_turno
ORDER BY
  cantidad_clases DESC
LIMIT 1;
```

Resto de consultas:

```
UPDATE actividades SET descripcion = %s, costo = %s WHERE id = %s
SELECT * FROM actividades
INSERT INTO clase (ci_instructor, id_actividad, id_turno, dictada) VALUES (%s, %s, %s, false)
UPDATE clase JOIN turnos ON clase.id_turno = turnos.id SET clase.ci_instructor = %s, clase.id_turno = %s WHERE clase.id = %s   AND CURRENT_TIME() > turnos.hora_fin
INSERT INTO alumno_clase (id_clase, ci_alumno, id_equipo) VALUES (%s, %s, %s)
DELETE FROM alumno_clase WHERE ci_alumno = %s AND id_clase = %s
```