

Sistema para **Gestión de Reserva** **de Salas de Estudio**

Integrantes: Thomas Cooper, Emanuel Fraga y Nicolas Vazquez

Docentes: Federico Valiño y Agustin Gonzalez

Introducción

El objetivo central del sistema es permitir la administración eficiente de salas de estudio por parte de la universidad, asegurando las distintas características:

- Tener el registro estructurado de salas, edificios, participantes y programas académicos;
- La realización de reservas con validaciones obligatorias;
- El control de asistencia a cada reserva;
- La aplicación automática de sanciones ante inasistencias reiteradas;
- La generación de reportes estadísticos basados en consultas SQL.

Se implementó el backend del sistema en Python(Flask) junto a una base de datos MySQL, manejando la lógica mediante SQL.

Planificación para cumplir la consigna

Planificamos el proyecto mediante la herramienta de proyectos del propio github y lo asociamos al repositorio. Establecimos dos metas, el avance y la entrega final para poder asociar las tareas a estas metas y saber gracias al porcentaje cuánto vamos y cuánto no falta.

Las tareas están bajo tres categorías: Documentación, Programa y Base de datos para gestionarlas mejor. Ya agregamos algunas tareas menores como la creación de la base de datos, la tabla, las distintas bitácoras, etc.

También, pusimos objetivos de aprendizaje para cumplir la consigna. Tuvimos que investigar más sobre Docker y Flask por ejemplo.

Modelo de datos

El modelo se diseñó a partir de las entidades y relaciones requeridas por la consigna, teniendo en cuenta las características de las tablas. Para ello, se tomaron distintas decisiones respecto a las tablas que deben estar creadas

- ***Participante***

Representa la unidad básica del sistema, que son los estudiantes y docentes que interactúan con las salas. Se utiliza la CI como clave primaria por ser un identificador único de cada individuo, además sólo contiene información sobre el participante y no de los roles o programas académicos de los que integra.

- ***Facultad***

Es donde se guarda la información de cada facultad, permitiendo organizar la información según la estructura de la universidad. La clave primaria es el id, el cual lo hicimos como un varchar y no como autoincremental para que sea más nemotécnico al momento de identificarla.

- ***Programa Académico***

En esta tabla se incluyen las carreras o posgrados, donde cada

programa pertenece solo a una única facultad. Se incluye el atributo tipo para poder diferenciar entre los programas de grado y posgrado, siendo útil para las validaciones posteriores que se hacen al agendar una sala.

- ***Participante_Programa_Académico***

Esta tabla está hecha para que un participante pueda tener varios programas, donde se incluye el atributo rol para validar las reglas de uso de las distintas salas. Se utiliza una clave primaria id_alumno para poder simplificar las operaciones y evitar duplicados.

- ***Edificio***

Agrupar las distintas salas por ubicación, además contiene toda la información necesaria para las validaciones.

- ***Sala***

En esta tabla se define cada espacio reservable, siendo la forma de identificarlas el nombre_sala + edificio, ya que podrían existir salas con el mismo nombre pero en distintos edificios. Además, tiene el atributo para guardar el tipo, para cumplir con las validaciones según el participante que vaya a reservar junto a la capacidad máxima.

- ***Turno***

Contiene los bloques horarios disponibles, permitiendo que sean reutilizables entre reservas y una fácil validación de solapamiento entre turno y los límites de reservas impuestos.

- ***Reserva***

Esta tabla es la que gestiona las reservas de las salas, siendo de las principales en cuanto al uso del sistema ya que depende de las demás. Posee un id único que lo hicimos autoincremental, además de tener el atributo estado que permite cumplir con las reglas del sistema y con el sistema de sanciones/reportes.

- ***Reserva_Participante***

Permite que una reserva tenga múltiples participantes, ya que cada sala puede tener varios de ellos. Es la tabla que incluye el atributo asistencia para registrar si un participante asistió o no, siendo un dato importante para la generación de sanciones y reportes.

- ***Sanción_Participante***

Indica períodos de sanción derivados de inasistencias. Cada sanción tiene fecha de inicio y fin, permitiendo hacer las validaciones temporales necesarias para reservar una sala.

Implementación del Backend

El sistema se implementó en un conjunto acotado de librerías y módulos basados en Python, siguiendo la siguiente estructura:

- app.py: reglas de negocio y direccionamiento
- validaciones.py: todas las validaciones necesarias y requisitos

- database.py: conexión y ejecución de SQL
- /templates: archivos HTML

Para una mejor funcionalidad del sistema, se usó el framework Flask para el desarrollo de la aplicación web. Se seleccionó el mismo ya que permite mantener un control sobre el flujo de la aplicación, las consultas SQL y la estructura del proyecto sin mayores restricciones adicionales, ya que permite controlar las consultas de manera directa y preservando la separación del backend con la base de datos, generando rutas claras para cada operación que se realiza.

En relación a HTML, se usó el motor de plantillas Jinja2 para poder tener una mejor organización a nivel de estructuras (usando por ejemplo una base.html para el header) y generando la claridad del código.

Dentro de database se utilizó el MySQL Connector, ya que nos permite ejecutar las consultas SQL directamente desde Python, permitiendo generar ejecuciones explícitas de insert, update y select sin tener que usar ningún ORM.

Reglas obligatorias

Para poder reservar se deben respetar ciertas reglas, las cuales se encuentran dentro de validaciones.py. Para eso, tenemos las siguientes implementaciones a nivel SQL:

Obtener info del usuario:

```
SELECT pa.tipo, ppa.rol
FROM participante_programa_academico ppa
JOIN programa_academico pa
    ON pa.nombre_programa = ppa.nombre_programa
WHERE ppa.ci_participante = %s
LIMIT 1;
```

Verificar si está sancionado:

```
SELECT COUNT(*) AS total
FROM sancion_participante
WHERE ci_participante = %s
AND fecha_inicio <= %s
AND fecha_fin >= %s;
```

Verificar si la sala está ocupada

```
SELECT COUNT(*) AS total
FROM reserva
WHERE nombre_sala = %s
```

```
AND edificio = %s
AND fecha = %s
AND id_turno = %s
AND estado = 'activa';
```

Verificar las horas reservas en el día:

```
SELECT COUNT(*) AS bloques
FROM reserva r
JOIN reserva_participante rp ON rp.id_reserva = r.id_reserva
WHERE rp.ci_participante = %s
AND r.fecha = %s
AND r.edificio = %s
AND r.estado = 'activa';
```

Verificar reservas activas:

```
SELECT COUNT(*) AS total
FROM reserva r
JOIN reserva_participante rp ON rp.id_reserva = r.id_reserva
WHERE rp.ci_participante = %s
AND YEARWEEK(r.fecha,1) = YEARWEEK(%s,1)
AND r.estado = 'activa';
```

Verificar si tiene solapamiento:

```
SELECT COUNT(*) AS total
FROM reserva r
JOIN reserva_participante rp ON rp.id_reserva = r.id_reserva
WHERE rp.ci_participante = %s
AND r.fecha = %s
AND r.id_turno = %s
AND r.estado = 'activa';
```

Verificar la sala al editar:

```
SELECT COUNT(*) AS total
FROM reserva
WHERE nombre_sala = %s
AND edificio = %s
AND fecha = %s
AND id_turno = %s
AND id_reserva <> %s
AND estado = 'activa';
```

Verificar el solapamiento en edición:

```
SELECT COUNT(*) AS total
FROM reserva r
JOIN reserva_participante rp ON rp.id_reserva = r.id_reserva
WHERE rp.ci_participante = %s
      AND r.fecha = %s
      AND r.id_turno = %s
      AND r.id_reserva <> %s  -- excluye la reserva editada
      AND r.estado = 'activa';
```

Verificar la capacidad:

```
SELECT capacidad
FROM sala
WHERE nombre_sala = %s AND edificio = %s
LIMIT 1
```

Implementación del Frontend

El sistema utiliza plantillas HTML que extiende de base.html, se tomó esta decisión para que el diseño sea unificado entre todas las secciones. Además, se usó un sistema de avisos mediante flash para poder mostrar cuando uno de los requisitos no se cumple.

Las plantillas desarrolladas en HTML fueron las siguientes:

- Participantes: alta, baja, modificación, listado
- Salas: alta, baja, modificación, listado
- Reservas: alta, listado, baja, modificación, control de asistencia
- Sanciones: Alta, listado de activas y vencidas, modificación, baja
- Reportes: consultas SQL elaboradas

Reportes

Todas las consultas fueron implementadas directamente con SQL, además se tomó la decisión de agregar las consultas de participantes con más faltas, ocupación promedio por turnos y salas con ocupación crítica.

Por el lado de los participantes con mayor cantidad de faltas, consideramos que es bueno incluirlo ya que el sistema tiene un mecanismo de control de asistencia y generación de sanciones, sin embargo no se muestra por defecto las inasistencias que tienen los participantes (sin llegar a ser sanciones). Esto puede llegar a ser útil en un sistema de reservas para identificar ciertos perfiles de usuarios con comportamientos no deseados o evaluar la efectividad de las sanciones.

En cuanto al porcentaje de ocupación por turno, aunque la consiga ya incluye reportes sobre salas y reservas, no se contempla el análisis temporal, el cual puede servir para conocer los horarios más utilizados y así que se

puedan tomar mejores decisiones respecto a la optimización de la disponibilidad de salas.

Por último, se agregó la consulta para saber las salas con ocupación crítica ya que es importante en un sistema de reservas, saber cuáles son las salas que generan una alta demanda; esto sirve para detectar posibles patrones de uso y priorizar las decisiones frente a la gestión de las salas.

Conclusión

En este obligatorio, pudimos aplicar los conocimientos que teníamos sobre SQL que venimos aprendiendo a lo largo del semestre. También, vemos más puntos positivos de hacer este obligatorio, ya que tuvimos que investigar sobre cómo usar Docker para usar contenedores y sobre Flask para el desarrollo de la aplicación web. Luego, como equipo nos sentimos cómodos al dividirnos las tareas y la comunicación fue clave para cumplir con la consigna.