Bases de Datos I

Sistema de reservas de salas de estudio

Docentes:

Juan Andrés Kosut

Sofía Guerrico

Estudiantes:

Natanael Fernández

Joaquín de Dios

María Victoria Riccetto

Fecha:

23 de noviembre de 2025

Introducción

El problema planteado por los docentes consiste en desarrollar una solución que permita automatizar la gestión de las salas de estudio, garantizando que se cumplan la normativa de la biblioteca de la UCU. Esta solución está diseñada para facilitar el trabajo de los administrativos, también se espera que se mejore la experiencia de estudiantes y profesores al momento de utilizar las salas y reservarlas. Para esto se implementó una basa de datos relacional en “Mysql”, un “backend” en Python y un “frontend” desarrollado en React con Vite, siguiendo todas las validaciones que se requieren.

En este informe se fundamentan las decisiones que tomamos, se incluyen las herramientas y se describen las mejoras que hemos aplicado al modelo de datos. Se espera que el resultado final de este proyecto constituya un primer acercamiento funcional hacia un sistema de gestión de reservas robusto que en un futuro pueda ser utilizado y ampliado por la universidad.

Justificación de Cambios

A lo largo del desarrollo de la base tuvimos que realizar varios cambios en su composición y estructura. Uno de los cambios más importantes fue que tuvimos que crear una nueva base de datos para este trabajo obligatorio, la base anterior nos estaba dando errores al momento de hacer operaciones como los inserts o updates, también cuando empezamos a utilizar github nos daba errores por eso la creacion de una nueva base de datos.

Luego de tener ya todo listo, las tablas con los datos insertados, hubo una tabla que la tuvimos que borrar y crear de nuevo, esta era la tabla reserva. En un principio nos estaba generado problema con consultas y modificaciones de datos que queríamos hacer. Pero, ese no era el único problema; habíamos insertado datos que no estaban relacionados con otras tablas, entonces al momento de consultar no eran devueltos de forma correcta. Luego de esto insertamos correctamente los datos necesarios para el funcionamiento de la tabla y sus consultas para que devolviera más de una reserva por sala en un mismo turno.

Otro cambio que hicimos fue en la tabla turno, esta tenía un dato fecha y el tipo de dato era “datetime”, lo cambiamos a tipo de dato “time” porque solo necesitábamos la hora, no toda la fecha exacta. Dividimos cada turno en un horario. Por ejemplo, el turno uno es de 8hs a 9hs y así sucesivamente. Luego de seleccionar la hora de reserva se elige uno de los turnos y una fecha, de esta forma se puede elegir un mismo turno en distintas fechas.

También, agregamos más datos a la tabla “sancion_participante” para que tenga datos suficientes para consultar. Al inicio tenía solo dos datos.

En la tabla participante modificamos el dato de asistencia, para que devolviera que un participante había asistido más de una vez a una sala. En participante programa académico también cambiamos en más de un dato a rol de docente, para que también devolviera de forma correcta.

A la hora de modificar la tabla reserva, no utilizamos “update” porque debíamos modificar varias tablas, lo más conveniente fue eliminar la tabla que queríamos modificar y crear una nueva, lo mismo hicimos para otras tablas.

Por último, en la tabla reserva_participante utilizamos una eliminación en cascada. Esta decisión fue tomada debido a que a la hora de eliminar una reserva no se nos permitía debido a que la tabla reserva_participante estaba usando el identificador de la reserva como “foreign key”, al intentar eliminar la “foreign key” el programa arroja un error. Por eso, decidimos modificar la tabla “reserva_participante” para que cuando borremos una reserva se elimine en esa tabla también.

Algunas consideraciones

Creemos que no es una buena práctica concatenar “strings” porque si se hiciera:

query = "INSERT INTO sala VALUES (" + nombre + ")"

Alguien podría hacer un ataque Sql injection, entonces implementamos el uso de %, MySql connector limpia automáticamente los valores, evitando así posibles ataques a nuestra base de datos.

Algunas de las validaciones que tuvimos que hacer al momento de trabajar con las salas de estudio. Las salas solo se pueden reservar por bloque de hora, no se puede hacer, por ejemplo, de 8:30 a 10, en ese caso se debe de reservar de 8 a 10. Tenemos tres tipos de salas, la primera sin ninguna restricción de rol es de uso libre, la segunda son exclusivas para estudiantes de grado y por último exclusivamente para docentes. Solo se pueden hacer tres reservas semanales y no se puede reservar por más de dos horas diarias, un aspecto que se debe tener en cuenta es que los estudiantes y profesores no tienen limitaciones al momento de reservar sus salas correspondientes.

Cada reserva tiene asociado los datos de todos los participantes que van a ocupar la sala, ya sean estudiantes o docentes, no se puede exceder la capacidad de la sala. El sistema debe registrar la asistencia de cada uno de los participantes, en el caso de que nadie se presente en el día y hora de la reserva, se les notificará y serán sancionados con dos meses sin poder realizar reservas.

Descripción de la solución

Nuestra base de datos está organizada bajo la estructura de modelo relacional que vimos en clase, compuesta por varias entidades principales relacionadas entre sí mediante la utilización de Primary Key y Foreign key. A continuación, se detallará su estructura.

1. Tablas principales
   1.1 Login : contiene el correo (pk) para iniciar sesión y contraseña.
   1.2 facultad: contiene el nombre de las facultades y su identificador único. Id_facultad (pk) y nombre
   1.3 programa_academico: contiene el nombre de cada programa y un identificador. Nombre_programa(pk) y id_facultad (esta es una foreign key que se relaciona con facultad.id_facultad), es la facultad a la que pertenece el programa académico.
   1.4 Participante: registra a los estudiantes que vayan a reservar una sala, cuenta con la cédula de identidad (pk), el nombre, apellido y email.
   1.5 participante_programa_academico: relaciona a los programas con los estudiantes que los cursan. Cuenta con un. Identificador único que es primary key(id_alumno_programa), con la cédula de cada participante (referencia a la ci de la tabla participante), identificador para el programa académico (esta es foreign key, referencia al identificador de la tabla programa_acdemico).
   1.6 edificio: representa a los edificios disponibles que cuentan con salas de estudio. Tiene como primary key id_edificio y luego tiene otros datos como nombre_edificio, dirección y departamento
   1.7 sala: representa las salas físicas disponibles para reservar. Cuenta con un identificador único que actúa como pk, también tiene un id_edificio (referencia a edificio.id_edificio), luego cuenta con otros datos como lo son nombre_sala, capacidad y tipo_sala.
   1.8 Turno: contiene un identificador del turno (pk), luego tiene datos de hora de inicio y hora de finalización.
   1.9 Reserva: tiene un identificador de reserva (pk), identificador de sala (actúa como fk y referencia a sala.id_sala), identificador de turno (actúa como fk y referencia a turno.id_turno), luego cuenta con un estado de la reserva y fecha en la que se realizó.
   1.10 Reserva_participante: vincula a los participantes con las reservas que hicieron. Tiene un pk compuesta por (ci_participante, id_reserva), luego tiene como fk a ci_participante que referencia a participante.ci y por último id_reserva referencia a reserva.id_reserva.
   1.11 Sancion_participante: tiene como pk un identificador para cada sanción, la ci_participante que referencia a participante.ci y actúa como fk, luego tiene otros datos que son la fecha de inicio y fin de la sanción.

El modelo aplica la utilización de claves primarias para evitar la repetición de datos y claves foráneas para garantizar la coherencia entra las entidades. Los vínculos se aseguran de que.

- No exista un programa sin su facultad correspondiente.
- Que no se pueda asociar un participante a un programa que no existe.
- No se puede registrar reservas en una sala inexistente
- No es posibles insertar en la tabla reserva_participante datos como id_reserva o ci si no existen previamente, esto para evitar un error de claves foráneas.

En resumen, nuestra base de datos intenta gestionas un sistema de reservas de salas de estudios por parte de estudiantes o profesores asociados a distintos programas académicos dentro de distintas facultades. Un estudiante o docente puede pertenecer a múltiples programas y puede estar asociado a múltiples reservas, en las cuales también pueden participar varias personas.

El “backend” está formado en su mayoría por “endpoints”, los cuales permiten que el “frontend” tenga funcionalidades relacionadas a la base de datos. En estos “endpoints” se encuentra también la lógica de la autenticación de usuarios.

El backend fue creado con una estructura sencilla y organizada, en el archivo app.py se encuentran todos los “endpoints”, incluyendo los de autenticación. Para la autenticación se decidió utilizar un token manual sobre un JWT, para que la integración con el resto de las partes del sistema fuera más amigable.

Los “endpoints” se crear de manera que fueran faciles de comprender y tratando de que fueran pocas líneas de código. Esto ayudó a que el “frontend” pudiera crear componentes individuales que ayudan a la navegación dentro de la página.

Mantener cada paso lo más compacto posible sirvió para que la conexión entre las tres partes fuera relativamente simple.

Tecnologías utilizadas

Para el desarrollo de este proyecto se utilizaron distintas herramientas que nos facilitaron el diseño, desarrollo, implementación y gestión de este, se enumeran a continuación.

- Git y GitHub: para trabajo colaborativo entre los integrantes del equipo, administración del repositorio y control de versiones.
- Datagrip: editor utilizado para escribir y organizar los archivos SQL del proyecto.

- MySQL: sistema de gestión de bases de datos relacional utilizado para la creación de tablas, manipulación de datos y ejecución de consultas SQL durante el desarrollo.
- Pyhton: para crear el “backend” y distintas funcionalidades requeridas.
- Flask: framework utilizado para facilitar la creación de endpoints.
- Postman: plataforma en la que probamos la API.
- React + Vite: para desarrollar el frontend de forma correcta.

Conclusión

El desarrollo del sistema de gestión de reservas de salas de estudio permitió transformar un proceso que anteriormente era realizado de forma manual en una solución centralizada, moderna y estructurada. A lo largo del proyecto se diseñó una base de datos relacional robusta, se siguieron la normativa propuesta por la universidad y se diseñó un “backend” funcional que garantiza el correcto, registro, control y administración de reservas, participantes y sanciones.

Este trabajo también nos permitió identificar mejoras posibles en el modelo de datos y en la experiencia de uso, esto demuestra que el sistema es escalable y que en un futuro puede ser mejorado para poder cubrir nuevas necesidades o requerimientos.