Informe obligatorio
Bases de Datos I

Martina Caetano
Mateo Gutiérrez

# INFORME

## Bitácora

### El viernes 18 de octubre

Nos reunimos presencialmente para comenzar con el obligatorio. Durante una hora organizamos el trabajo para luego empezar con la creación de las tablas brindadas en la propuesta.

Sobre la creación de las tablas decidimos que:

A cada ID le asignamos autoincrement para no tener que inventar un número cada vez que se agregara un curso, etc. Decidimos usar el valor INT en costos puesto que un Uruguay los precios no son decimales.

No modificamos por ahora las primary key, no agregamos tablas ni columnas.

Para asignar las foreign keys a cada tabla, nos fijamos en cuáles de los datos que necesitaban ya existían en otra, para poder tomarlos de ahí en lugar de completar un nuevo campo con la misma información. En el caso de *equipamientos, id_actividad* es clave foránea tomada de *actividades,* puesto que el id se autoincrementa cada vez que se agrega una nueva a la tabla. Lo mismo sucede con *clase* y *alumno_clase*; la primera toma *id_actividad* de actividades e *id_turno* de turnos, y la segunda toma *id_clase* de *clase* e *id_equipamiento* de *equipamiento*. *Clase* también toma *ci_instructor* de *instructores* ¸ para no tener que reingresar este valor, al igual que *alumno_clase* toma *ci_alumno* de *alumnos*.

Para la columna *dictada*, decidimos usar boolean siendo true dictada y false que no fue dictada, asignamos false por defecto para que las clases comiencen siendo no dictadas.

### El miércoles 23 de octubre

Agregamos la tabla rol con los tres roles posibles, administrador, instructor y alumno y un id para identificarlos.

A la tabla usuario le agregamos la columna id_rol con una clave foránea hacia el id de la tabla rol.

Comenzamos a investigar que opciones tenemos para el frontend ya que ninguno de los dos cursó diseño web.

Realizamos las inserciones previas en las tablas rol, equipamiento, turnos, actividades y las subimos al repositorio en el archivo Inserciones-previas.sql.

Martina Caetano
Mateo Gutiérrez

En la tabla rol agregamos lo previamente dicho, en las demás agregamos los elementos que menciona la letra.

Para que dos instructores no puedan dar clases distintas a la misma hora agregamos una contraint unique entre ci_instructor e id_turno en la tabla clase.

**Viernes 25 de octubre**

Creamos los usuarios de la base de datos, estos son: administrador, instructor y alumno.

Luego decidimos que permisos darla a cada rol sobre las tablas.

Tabla login:

El administrador tiene todos los permisos.

Los usuarios alumno e instructor solamente pueden hacer select sobre la columna correo.

Actividades:

El administrador tiene todos los permisos.

Los usuarios alumno e instructor pueden hacer select.

Equipamiento:

El administrador tiene todos los permisos.

Los usuarios alumno e instructor pueden hacer select.

Instructores:

El administrador tiene todos los permisos.

El usuario alumno tiene permiso de select.

Instructor puede hacer select, insert y delete, puede modificar las columnas nombra, apellido y correo.

Turnos:

El administrador tiene todos los permisos.

Los usuarios alumno e instructor pueden hacer select.

Alumnos:

Administrador y alumno tienen todos los permisos.

Martina Caetano
Mateo Gutiérrez

Instructor puede hacer select.

Clase:

Todos los permisos para el administrador.

Alumno solo puede hacer select.

Los instructores pueden hacer select, tambien pueden hacer update en las columnas id y dictada.

Alumno_clase:

El admin tiene todos los permisos.

Instructor puede hacer select.

El usuario alumno tiene el permiso de select, insert y delete.

Rol:

El admin tiene todos los permisos.

**Viernes 1 de noviembre**

Comenzamos con el backend, implementamos la función obtener_rol que devuelve el rol del correo pasado por parámetro y la función conectarse que realiza la conexión a la base de datos con el usuario que recibe por parámetro.

También declaramos funciones en el backend para agregar, eliminar y modificar alumnos, docentes y turnos.

Continuamos investigando y consultando acerca de cómo hacer el frontend.

**Jueves 7 y 8 de noviembre**

Continuamos indagando acerca de cómo realizar el frontend, optamos por intentar utilizar React.

Inicialmente buscamos información en internet sobre cómo funciona React, vimos algunos tutoriales y comenzamos con la pantalla de inicio. Luego de dos reuniones apenas logramos tener un menú de inicio de sesión que además no funcionaba correctamente. Como cada vez queda menos tiempo, finalmente

Martina Caetano
Mateo Gutiérrez

decidimos desistir y optamos por solamente utilizar la consola de PyCharm.

**Martes 12 de noviembre**

Decidimos establecer varias pantallas, siendo la principal una con tres opciones: iniciar sesión, registrarse y salir del programa. En caso de iniciar sesión como alumno, se muestran las opciones para el alumno, lo mismo si se inicia como administrador o instructor. Para cada caso se muestra un menú personalizado con las opciones que correspondan.

Acordamos que para el menú del alumno se mostrarían 5 opciones, una para ver y modificar sus datos personales, otra para inscribirse a clases, una tercera opción para ver sus clases donde una vez dentro pueda decidir si darse de baja de alguna, también podrá darse de baja del sistema y cerrar sesión.

Los instructores también podrán ver sus datos, podrán elegir una clase, ingresar en un menú para ver sus clases donde pueda marcarlas como dictadas o darse de baja de estas y por último también podrán darse de baja del sistema y cerrar sesión.

En cuanto al usuario administrador decidimos que podrá ingresar a un menú de ABM de turnos, podrá modificar las clases ya existentes, siempre y cuando no estén en curso, y podrá cerrar su sesión.

Además, a cada uno de los usuarios le daremos acceso a los reportes de la UCU sobre las actividades, pero solamente los administradores podrán ver el registro de los ingresos.

Luego de tener ambos en claro como funcionaría el programa nos dividimos tareas para comenzar a dedicarnos por completo en el programa en python.

**Viernes 15 de noviembre**

Ya teniendo un primer código de los menús de alumno, instructor y administrador. Comenzamos a trabajar en el flujo principal del programa. Realizamos el main que básicamente llama a inicio_sesión y guarda el resultado, el correo del usuario, en una variable, se llama a la función obtener_rol y según lo que devuelve es el menú al que se dirige el programa para mostrarle al usuario las opciones correspondientes.

**Sábado 16 de noviembre**

Martina Caetano
Mateo Gutiérrez

Encontramos una librería para generar tablas que nos sirven para simular los menús de opciones y como se muestran las diferentes consultas, además tienen bastante mejor apariencia que una simple lista de opciones. Dicha librería se llama tabulate. Además, utilizamos la librería time para agregar espera entre la impresión de las tablas y mensajes para que sea más fácil para la lectura del usuario.

**Lunes 18 de noviembre**

Importamos la librería hash para no guardar las contraseñas directamente como la escribe el usuario, además implementamos una función que verifica que al intentar iniciar sesión el hash de la contraseña de ingreso coincida con el hash de la base de datos.

Agregamos la restricción de que un alumno no pueda estar inscripto a distintas clases en el mismo horario, para esto realizamos una consulta por sus clases actuales y comparamos las horas de inicio y fin con la hora actual.

Nos percatamos de un problema y es que debido a la contraint unique que agregamos a la tabla clase para que no se repita la dupla (ci_instructor, id_turno) ocurre que si un instructor marca como dictada una clase luego no puede volver a tomar un curso en ese horario por la condición mencionada ya que la dupla (ci_instructor, id_turno) se repetiría, pero esto no tiene sentido ya que el instructor ya no está dando la clase que está marcada como dictada. Para solucionar esto decidimos eliminar esa condición de la tabla y manejar estos casos directamente en el programa.

**Martes 19 de noviembre**

A las restricciones de los horarios le realizamos una modificación para que no solo se chequee que no coincidan los turnos, sino que además se evite cualquier solapamiento, por ejemplo, si un alumno o instructor está anotado a una clase de 12:00 a 14:00 no se pueda anotar a otra de 13:00 a 15:00 por ejemplo.

Para facilitarnos las comparaciones de fechas importamos las operaciones datetime y timedelta.

Martina Caetano
Mateo Gutiérrez

Entre el **miércoles 20 y el viernes 22 de noviembre** nos dedicamos a buscar y corregir pequeños errores del programa.

## Detalle de la lógica del programa

**Funciones del archivo funciones.py:**

obtener_rol(correo): devuelve el rol del usuario pasado por parámetro.

obtener_fecha_nacimiento(): le pide una fecha al usuario y válida el formato.

validar_hora(): valida que una hora este en formato HH:MM.

coincidencia_turno(turnos, hora_inicio_dt, hora_fin_dt): checkea que no haya ningún turno en la lista turnos que se solape con las horas de inicio y fin.

**conexion.py:**

conectarse(usuario): obtiene el rol del usuario pasado por parámetro y se conecta a la base de datos con las credenciales correspondientes al rol.

**hash.py:**

hash_password(password): hashea la contraseña pasada por parámetro.

check_password(password, hashed): checkea que el hash de password coincida con el guardado en la tabla login.

**inicio_sesion.py:**

inicio_sesion(): muestra el menú de inicio de sesión y maneja el flujo de las tres opciones posibles, si se trata de un inicio de sesión le pide al usuario sus datos y checkea que sean correctos. Para un nuevo usuario le pide correo, contraseña y rol, si es administrador o instructor le informa al usuario que se le envió un mail (ficticio) para validar su identidad, luego llama a insert_ login y a alta_usuario en caso de no ser un administrador. Devuelve el correo en caso de darse el inicio de sesión exitosamente.

**login.py**

insert_login(correo, contraseña, id_rol): hace el insert a la tabla login con los datos del nuevo usuario.

alta_usuario(correo, contraseña, id_rol): le pide al usuario que ingrese sus datos y realiza el insert en la tabla alumno o instructor dependiendo de cuál sea el id_rol.

Martina Caetano
Mateo Gutiérrez

baja_login(correo): elimina de la tabla login el usuario con el correo pasado por parámetro.

validar_credenciales(correo, contraseña): checkea que exista un usuario con el correo y llama a check_password para validar que la contraseña sea correcta.

usuario_existente(correo): devuelve True si ya hay un usuario en el sistema dueño del correo del parámetro.

**menu_admin.py:**

menu_admin(correo): muestra las opciones de un administrador y le pide que ingrese la acción que desea realizar, luego llama a la función que se encargue de lo seleccionado.

menu_admin_turnos(): muestra las opciones de ABM de turnos y le pide al usuario que elija una.

**menu_instructor.py:**

menu_instructor(correo): muestra las opciones de un instructor y le pide que ingrese la acción que desea realizar, luego llama a la función que se encargue de lo seleccionado.

**menu_alumno.py**

menu_alumno(correo): muestra las opciones de un alumno y le pide que ingrese la acción que desea realizar, luego llama a la función que se encargue de lo seleccionado.

**alumnos.py:**

baja_alumno(correo): realiza los delete de las filas de las tablas alumno_clase y clase que se vinculen con el alumno que tengo el correo pasado por parámetro.

modificar_alumno(correo): solicita al alumno que ingrese sus nuevos datos y llama a update_alumnos.

update_alumno(nombre, apellido, fecha_nacimiento, telefono, correo): hace el update en la tabla alumnos con los datos pasados como parámetro.

select_alumno(correo): devuelve todos los datos del alumno.

info_alumno(correo): muestra los datos del alumno, y le pregunta si desea modificarlos, de ser así llama a modificar_alumno.

**alumno_clase.py:**

Martina Caetano
Mateo Gutiérrez

alta_alumno_clase(correo): le muestra al alumno las clases y se le solicita el id de la clase a la que quiere inscribirse. Checkea que no esté inscripto ya a esa clase y que no genera una superposición con las clases actuales del alumno. Luego muestra los equipamientos disponibles para la actividad seleccionada y llama a insert_alumno_clase con los datos elegidos por el alumno.

insert_alumno_clase(id_clase, ci_alumno, id_equipamiento): realiza el insert en la tabla alumno_clase.

mostrar_alumno_clases(correo): le muestra al alumno sus clases y le pregunta si desea darse de baja de alguna, en ese caso llama a baja_alumno.

baja_alumno_clase(correo, id_clase): realiza el delete de la tabla alumno_clase.

validacion_ci(correo, id_clase): valida que exista una clase con el id id_clase

**instructor.py:**

insert_instructor(ci, nombre, apellido, correo): inserta en la tabla instructores.

baja_instructor(correo): hace el delete de la tabla instructor luego de haber eliminado todas las filas que hagan referencia a él en las tablas alumno_clase y alumno.

modificar_instructor(correo): pide al instructor que ingrese sus nuevos datos y se los pasa a la función update_instructor.

update_instructor(nombre, apellido, correo): hace la modificación en la tabla instructores.

obtener_instructor(correo): devuelve el instructor que porte el correo pasado por parámetro.

info_instructor(correo): muestra al instructor sus datos.

**clase.py:**

obtener_clases(correo): devuelve todas las clases.

alta_clase(correo): muestra las actividades al instructor y le pide que seleccione alguna, luego muestra los turnos y checkea que no exista problemas de superposición entre los turnos de las demás clases del instructor y su nueva elección. Por último, realiza el insert en la tabla clase.

mostrar_clases_instructor(correo): muestra las clases del instructor y le da la opción de dejar de enseñar o marcar como dictada una clase.

Martina Caetano
Mateo Gutiérrez

modificar_clase(): le muestra al admin todas las clases y le pregunta cual desea modificar, checkea que la clase no esté en curso. Luego da la opción de elegir el nuevo instructor y turno de la clase, revisa que no exista una superposición con los horarios del instructor. Llama a update clase.

update_clase(id_clase, ci_instructor, id_turno): realiza la modificación de la tabla clase.

**turno.py:**

agregar_turno(): muestra los turnos actuales, le pide al administrador el horario de inicio y fin en formato HH:MM. Checkea que no exista ese turno y que la duración sea de 2 horas. Llama a insert_turno.

insert_turno(hora_inicio, hora_fin): realiza el insert en la tabla turnos.

eliminar_turno(): muestra los turnos y pregunta cuál se desea eliminar. Si se dictan clases en el horario seleccionado se le informa al administrador y se solicita una confirmación, luego llama a delete_turno.

delete_turno(id): elimina el turno de la tabla turnos.

modificar_turno(): muestra los turnos y pregunta el id del que quiere modificar, si se dictan clases en ese horario se pide confirmación para editarlo. Llama a update_turno.

update_turno(id, nueva_hora_inicio, nueva_hora_fin): realiza el update.

obtener_turnos(correo): devuelve todos los datos de la tabla turnos

clases_por_turno(id_turno): devuelve las clases que se dictan en el turno pasado por parámetro.

**actividades.py:**

modificar_actividades(): muestra las actividades y le pide al administrador que elija cuál quiere modificar, le pide que ingrese el nuevo costo de la actividad y llama a update_actividad.

update_actividad(id, costo): hace el update en la tabla actividad.

obtener_actividades(correo): devuelve todas las actividades.

**reportes.py:**

mostrar_reportes(correo): muestra los reportes de la universidad, en caso de tratarse del correo de un administrador también se muestra el reporte de los ingresos por actividad.

Martina Caetano
Mateo Gutiérrez

**main.py:**

Llama a inicio_sesion y guarda el resultado en una variale llamada correo, luego dirige el flujo del programa a menu_admin, menu_instructor o menu_alumno dependiendo del resultado de obtener_rol(correo).